AD-A104 147 MITRE CORP MCLEAN VA METREK DIV F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-1 FAA-RD-81-45-1 NL

1 OF 5
AD
A104147

LEVEL II

12

DOT/FAA/RD-81/45, 1
MTR-81W120, 1

Automatic Traffic Advisory and Resolution Service (ATARS) Algorithms Including Resolution-Advisory-Register Logic

R.H. Lentz, W.D. Love, T.L. Signore
R.A. Tornese, A.D. Zeitlin

The MITRE Corporation
Metrek Division
McLean, Virginia 22102

June 1981

Volume 1-Sections 1 through 11

US Department of Transportation
Federal Aviation Administration
Systems Research & Development Service
Washington, D.C. 20590

Technical Report Documentation Page

1. Report No. FAA-RD-81-45-1	2. Government Accession No. AD-A104 147	3. Recipient's Catalog No.
4. Title and Subtitle Automatic Traffic Advisory and Resolution Service (ATARS) Algorithms Including Resolution-Advisory-Register Logic		5. Report Date June 1981
		6. Performing Organization Code
7. Author(s) R.H. Lentz, W.D. Love, T.L. Signore, R.A. Tornese, A.D. Zeitlin		8. Performing Organization Report No. MTR-81W120-1
9. Performing Organization Name and Address The MITRE Corporation Metrek Division 1820 Dolley Madison Boulevard McLean, Virginia 22102		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. DOT-FA80WA-4370
12. Sponsoring Agency Name and Address U. S. Department of Transportation Federal Aviation Administration Systems Research and Development Service Washington, D.C. 20590		13. Type of Report and Period Covered
		14. Sponsoring Agency Code ARD-240

15. Supplementary Notes

16. Abstract

This document presents detailed computer algorithms for programming the Automatic Traffic Advisory and Resolution Service (ATARS). A major feature of this version of the ATARS algorithms is the capability to exchange resolution advisory information via the airborne Resolution Advisory Register (RAR). This provides for coordination of resolution advisories between ATARS and airborne collision avoidance systems and between adjacent ATARS sites in the absence of ground communication lines. The ground based ATARS computers use the surveillance data from the Discrete Address Beacon System (DABS) to provide properly equipped aircraft with traffic advisories and collision resolution advisories. These advisories are discretely delivered to the aircraft via the DABS data link. The ATARS algorithms are presented in two volumes rather than one large document in order to provide the algorithms in a more manageable form.

17. Key Words Collision Avoidance, Aircraft Separation Assurance, Traffic Advisory Service.	18. Distribution Statement Document is available to the public through the National Technical Information Service, Springfield, Virginia 22161

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages	22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

411

ACKNOWLEDGEMENT

Many people have contributed to the design, testing, and determination of parameter values which has culminated in the algorithms contained in this document. The personnel of the FAA's Systems Research and Development Service (ARD-241, ATARS Section) have provided many helpful suggestions to the testing and documentation of ATARS. Dr. John Scardina of ARD-241 has contributed to the design of the resolution model validation logic and the aircraft encounter data analysis chart contained in this document. Additional personnel of The MITRE Corporation have supported the ATARS development effort on a continuing basis for the past several years. Mr. Henri Bulterman developed the pseudocode language used in this document and provided Appendix C. Mr. Bulterman also developed programming tools which greatly facilitated the development of pseudocode for this document and participated in the testing of the resolution and detection logic. Ms. Nancy Fredman contributed to the testing and parameter value selection for the tracker turn sensing and vertical rate determination logic. Mr. Andrew Zeitlin served a dual role as author of several sections of this document and as having prime responsibility for the integration of all elements of the ATARS logic. Mr. Donald Roberts contributed to the final design review and editing of this document as well as having overall project and technical management responsibility for ATARS.

The following individuals were responsible for developing the logic design, specifying the required pseudocode, and authoring the cited full and partial section numbers in Volumes I and II:

Section No.	Author
3, 4, 5.1, 6.1	R. H. Lentz
6.3, 13.2, 15	W. D. Love
7, 8, 11, 19	T. L. Signore
12, 13	R. A. Tornese
1, 2, 5.2, 6.2, 9, 10, 14, 16, 17, 18	A. D. Zeitlin

TABLE OF CONTENTS

VOLUME I

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS
(Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS
(Concluded)

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to specify the computer algorithms and operation of the Automatic Traffic Advisory and Resolution Service (ATARS) and its interfaces with the Discrete Address Beacon System (DABS) and the Air Traffic Control System (ATC). Reference 1 describes the DABS sensor which provides surveillance data and ground-to-air and ground-to-ground communications to permit operation of the ATARS function.

This document provides algorithm specifications for the following:

1. Report processing and tracking logic
2. Conflict detection and resolution logic
3. A traffic advisory service
4. Logic to permit operation in a multi-site environment
5. Logic to interface ATARS with the Beacon Collision Avoidance System (BCAS)
6. Logic to treat various failure conditions

It does not, however, specify procedures to be followed by pilots or controllers using ATARS or standards to be met in supplying the ATARS function. The subject of standards is treated in Reference 1 and the ATARS function to be supplied is subject to all of the requirements of Reference 1 as if this document were incorporated in total at the point of reference.

Reference 2, which provides a broad conceptual-level description of ATARS, is a useful document for describing the philosophy and goals of the ATARS function in detail.

Reference 3 provides a detailed description of the DABS/ATARS function in the context of the ATC operational environment.

1.2 Definitions

Certain conventions and definitions of terms used in writing this document need to be explained.

Three terms are used in discussing the facilities at a particular location. The term "sensor" means the complete DABS sensor as described in Reference 1. The term "ATARS function" refers to all of the additional hardware and software required at a location to provide ATARS service. The ATARS function is described by this document. In this document, the term "ATARS function" is frequently used as if it were describing a separate piece of physical equipment. However, the implementation of ATARS may not be physically separate and so this term refers to a conceptually separate function or task. The third term, "site," refers to the DABS sensor and ATARS function at a single location, collectively. Any of these three terms may be qualified by the terms local and remote. Local refers to an item at the single location of principal concern. Remote refers to an item at any other location.

The term advisory is used to refer to a message to be delivered to an aircraft. There are several types of advisory messages generated by ATARS.

The term scan refers to the act of the sensor antenna rotating through one complete revolution, or to the time required for this act to take place.

Several terms are used to describe the DABS and collision avoidance avionics equipage of an aircraft and the distinctions between these terms need to be understood. An aircraft can be classified as radar-only (non-beacon) or as ATCRBS (Air Traffic Control Radar Beacon System) or DABS depending on the type of beacon transponder carried by the aircraft. An aircraft can also be classified as either ATARS equipped or unequipped depending on whether or not that aircraft has an ATARS display. This classification is used to select appropriate collision avoidance advisories for a given pair of aircraft. An ATCRBS aircraft and radar-only aircraft are always designated as unequipped. However, a DABS aircraft may be either ATARS equipped, BCAS and ATARS equipped, or neither (unequipped).

The term non-mode C is used to refer to a transponder-equipped aircraft (either ATCRBS or DABS) without mode C altitude reports, or an aircraft without a transponder (i.e., radar-only). The terms ATCRBS and DABS in this document normally imply mode C altitude is present.

1.3 Organization

Section 2 is a brief overview of the ATARS concept describing the services provided by the ATARS algorithms. This section is for familiarization only and is not a complete system description.

Section 3 provides a high-level view of the operation of the ATARS elements and discusses the coordination between them. It describes the external interfaces of the ATARS function. It also contains definitions of the data structures common to all ATARS processing tasks.

Section 4 provides a description and pseudocode for the report processing and tracking tasks.

Sections 5 through 16 contain descriptions and pseudocode for the message processing, aircraft processing, conflict detection, conflict resolution, BCAS coordination, and multi-site coordination functions of the ATARS system.

Section 17 provides pseudocode and a description of the algorithms to be implemented under various failure conditions of the DABS/ATARS system.

Section 18 contains a functional specification for the ATARS Status Monitoring and Reporting Function. Although this function must interface with many ATARS tasks and data structures, its requirements are collected in this section alone.

Section 19 presents the specification for the ATARS Data Extraction Function. This function also must interface with many tasks and data structures but is described in this section alone.

Appendix A collects all of the ATARS system variables and parameters, and presents nominal values for each parameter. This Appendix provides a cross-reference to the pseudocode in each section of the document.

Appendix B defines symbolic constants used in the pseudocode.

Appendix C defines the syntax for the pseudocode used in this document to define the ATARS algorithms.

Appendix D lists the references.

1.4 Pseudocode

The last part of Sections 4 through 17 contain the pseudocode for each of the tasks listed in Section 3. This approach to software specification has been adopted because it provides a clear, English-like specification, which is believed superior to flowcharts for conveying complex logic to the reader; and at the same time provides a precise software specification using a formal structure, which is again superior to flowcharts for error-free design. This document achieves these dual goals by using both a "high-level" English-like pseudocode and a "low-level" variable specification pseudocode, always printed side-by-side on facing pages. Each task, routine, or process is limited to one page, with no limit to the number of nested processes used to express the algorithm.

Each pseudocode section begins with its own Table of Contents. Next follows the declaration of any parameters and variables which are local to that task. The pseudocode for the task is next, followed by its first-level processes, in the order of their invocation, and then all lower-level processes in alphabetical order. Any routines or additional tasks then follow, with same content for each. Since this is pseudocode, and not actual machine code, even the low-level pseudocode contains English statements. No attempt has been made to produce finished code, but only to convey clear meaning to the programmer. This is frequently done where the final code is heavily implementation-dependent. For example, numeric values have not been assigned to system constants (Appendix B), in which the values used are not significant, so long as alternative values are unique. An explanation of the pseudocode is given in Appendix C. This should be read before beginning the pseudocode sections.

Within the pseudocode, certain abbreviations are frequently used in order to maintain the critical indentation which describes the logic, and limit each task or process to a single page. For example, "aircraft" is abbreviated as AC, and major data structures such as Pair Record, Conflict Table, etc. are sometimes abbreviated. References to system data structures are normally given in full, with both the structure name and the member name given. However, certain shorthand conventions are used at times, where the meaning should be clear to the programmer. These conventions are described at the start of the pseudocode section where they are used. A non-qualified variable name (i.e., structure not stated) refers to a variable which is local to the task in which it is used.

1.5 Features New to This Specification

ATARS has undergone considerable revision and refinement throughout its development, the most recent previous version being Reference 4. The major revisions new to this document are:

1. Resolution advisory coordination using the Resolution Advisory Register (RAR)

2. A Traffic Advisory Service providing multiple levels of service

3. Traffic Advisory and Domino provisions for non-mode C aircraft

4. A level-occupancy nonlinear vertical tracker

5. Various new and revised features in the Resolution Advisories Evaluation Routine

6. A Status Monitoring and Reporting function

7. A Data Extraction function

2. SYSTEM OVERVIEW

The basic concept of ATARS is very briefly reviewed here as background to the technical description of the algorithms. A complete functional description is presented in Reference 2. The discussion here is only intended to introduce ATARS to the program designer.

2.1 Summary Concept Description

The Automatic Traffic Advisory and Resolution Service is a ground based collision avoidance system to be implemented in the following environment:

1. Full x, y, and z (altitude reporting) surveillance or non-mode C (mode A or primary radar data) on all aircraft in the ATARS surveillance area

2. Direct digital data link to displays in the cockpits of aircraft receiving ATARS service

3. Aircraft with an operational Beacon Collision Avoidance System (BCAS) (see Reference 5)

4. Netted and non-netted adjacent DABS sites

5. An automated decision process

The Discrete Address Beacon System (DABS) provides the fully automatic surveillance and data-link communications capabilities which are prerequisite to the realization of ATARS.

The ATARS system monitors the location, altitude, and velocity of all aircraft throughout a contiguous airspace via the surveillance capability. A ground based computer processes the data and continuously provides proximity warning information and, when necessary, resolution advisories to aircraft receiving ATARS service. A limited traffic advisory service is provided to inform ATARS equipped aircraft of nearby non-mode C aircraft. Certain messages are generated by ATARS and displayed to the responsible air traffic controller at the ATC facility when a conflict involving a controlled aircraft is detected by the ATARS system.

2.2 Types of Encounters

The ATARS system behaves differently depending on whether the aircraft in conflict are under control of the ATC system (controlled aircraft) or not (uncontrolled aircraft), and on whether one aircraft is unequipped to receive ATARS resolution advisories or has not adequately responded to the original resolution advisory. The types of messages to be sent to the aircraft and the parameters used in the detection algorithms vary with the type of encounter involved.

2.2.1 Uncontrolled/Uncontrolled Encounters

ATARS is a limited form of ground based air traffic control which provides proximity warning and separation services to uncontrolled aircraft in a given region of airspace. It is intermittent in that it intervenes into the Visual Flight Rule (VFR) flight regime only when that flight's present course and altitude put it in proximity to, or in potential conflict with other traffic. It does not require the pilot to file a flight plan or to operate under an ATC clearance.

The look-ahead times and minimum miss distance criteria used to issue resolution advisories in uncontrolled/uncontrolled conflicts are of a "tactical" nature (e.g., 30 seconds and 0.5 nmi) and imply collision avoidance intervention only when a conflict is imminent. The uncontrolled aircraft still operates in a primarily "free flight" mode.

2.2.2 Uncontrolled/Controlled Encounters

In an uncontrolled/controlled encounter, the air traffic controller becomes another element in the resolution of a conflict. The sequence of events is as follows. When a pair of aircraft is observed to be on the order of 40 seconds from a violation of 1.2 nmi horizontal separation or 375 feet vertical separation, a Controller Alert Message is generated and displayed to the controller with responsibility for the controlled aircraft. This message will contain the conflict resolution advisory which ATARS would deliver to the uncontrolled aircraft if it were to issue one at this time. (If the conflict alert messages which are generated within the en route or terminal automation systems are already displayed for this aircraft pair, the controller message generated by ATARS need not be displayed in duplicate.) At the same time a threat advisory is issued to both pilots indicating the potential conflict and that they should attempt to visually acquire the threat aircraft and make a threat assessment. The controller observes the warning on his display and may elect to maneuver the controlled aircraft to avoid the uncontrolled aircraft or simply issue an advisory on the traffic. If no action is taken, at about 15 seconds later a resolution advisory is issued to the uncontrolled pilot informing him that he should perform the evasive maneuver indicated. If ATARS determines that the conflict situation has continued to deteriorate (approximately 25 seconds before closest approach) then the controlled aircraft is also issued a resolution advisory. Both of these resolution advisories are made available to ATC for display to the controller.

2.2.3 Controlled/Controlled Encounters

ATARS can serve a role as a backup collision avoidance or separation assurance system for conflicts between two controlled aircraft. It is not a controller automation aid as are the present conflict alert functions in the terminal and en route automation systems. ATARS is designed to maneuver controlled aircraft only when absolutely necessary, and is not intended to supplant the ATC system or routinely issue resolution advisories to controlled aircraft.

An ATARS Controller Alert Message is generated by ATARS at a time comparable to the ATC Conflict Alert time for display to the responsible controller. This message contains the conflict resolution advisories for both aircraft which ATARS would issue if it were to do so at that time. Both pilots are informed of the potential threat. If the conflict continues to deteriorate, ATARS will issue resolution advisories to the pilots approximately 25 seconds before closest approach is reached. These advisories are also made available to ATC for display to the controller.

The continued execution of ATARS with direct data link advisories to controlled aircraft as well as to uncontrolled aircraft can be a significant safety backup during periods of ATC hardware or software failure.

2.2.4 Encounters With More Than Two Aircraft

Logic has been developed to resolve conflicts involving more than two aircraft. Details of this logic are presented in a later section.

2.2.5 Encounters With One Aircraft Unequipped

The ATARS system can detect conflicts between one equipped aircraft and one aircraft which is unequipped. The system uses larger time thresholds so that the conflict can be satisfactorily resolved by issuing resolution advisories only to the equipped aircraft.

2.2.6 Encounters Which Are Not Resolved With Initial Resolution Advisories

Special logic to alter the resolution advisories is implemented in encounters which continue to deteriorate after initial advisories have been issued or in which additional significant information becomes available on the aircraft's maneuvering status. When such an encounter is detected, additional advisories may be issued to one or both aircraft.

2.3 Uplink Messages to be Sent to Aircraft

ATARS information is uplinked to equipped aircraft in display independent formats which are capable of supporting a wide range of display implementations. In order to facilitate the implementation of low cost avionics as well as to provide sufficient information to drive sophisticated avionics (such as a graphic display) without transmitting a large amount of unwanted information to some aircraft, ATARS has been designed to provide up to 8 classes of service.

This document describes 3 classes of service; classes 0, 1 and 2. Each class of service provides the full set of ATARS advisories; however, they differ in terms of the quantity of information provided.

Upon entry into ATARS coverage, each avionics installation will notify the ground of the class of service it desires. This will be the lowest class of service that will permit full functioning of the avionics installation, thereby making efficient use of the DABS data link.

2.3.1 Proximity Advisory

A proximity advisory is used to inform a pilot of the presence of nearby proximate, non-threatening aircraft. The message contains sufficient information to enhance visual acquisition. The horizontal range and relative altitude are used to determine when a proximity advisory will be issued. The horizontal range varies with the speed of the aircraft involved. If one aircraft receives an advisory because of the proximity of a second aircraft, that second aircraft (if ATARS equipped) will also be issued an appropriate proximity advisory indicating the presence of the first aircraft. Note that for ATARS service, at least one aircraft must be ATARS equipped.

2.3.2 Threat Advisory

When an aircraft is in potential conflict with another aircraft as determined by horizontal, vertical, and miss-distance tests, then a threat advisory is issued to warn the pilot of the potential collision situation. This message is given approximately 15 seconds in advance of the resolution advisory to give the pilots involved time to assess the situation on their own by locating each other visually using the relative bearing, altitude, and heading data from the threat advisory.

2.3.3 Resolution Advisory

The pilot will be given a negative resolution advisory either (1) to prevent a maneuver that, if executed, would cause a positive resolution advisory to be issued, or (2) when stopping an aircraft's vertical rate or rate of turn would produce sufficient separation to resolve the conflict. These advisories are in the form of generic "don't" messages (don't turn left, don't climb, etc.). All data provided for a threat advisory is transmitted with a resolution advisory. The vertical speed limit (VSL) is a negative advisory which requires that the pilot limit his rate of climb or descent.

A positive resolution advisory will be issued whenever ATARS determines, based upon a projection of the aircraft, that the aircraft will come closer than a specified separation threshold.

Resolution is accomplished by selecting the best maneuver for each aircraft for the particular geometry such that clearance of the hazardous condition will be provided. This is accomplished by modeling each aircraft as responding to all possible maneuvers and selecting the one which provides the most acceptable maneuver based on the consideration of many factors. The advisories are removed when the aircraft no longer satisfy the detection criteria for such advisories.

2.3.4 Own Message

The ATARS ground based system will provide periodic Own Messages to suitably equipped aircraft. This message contains information pertaining to own aircraft's tracked heading, ground speed, altitude, and turn rate as seen by ATARS. This information is used by the aircraft's on board display processor to aid in the presentation of ATARS generated advisories.

2.3.5 Terrain, Airspace, and Obstacle Avoidance Messages

ATARS will give an alert to an aircraft too near the terrain or an obstacle. A map of the terrain in the ATARS service area will be generated from U. S. Geological Survey Data and stored in the ATARS data base. Also, ATARS will provide an alert to pilots when a violation of restricted airspace is imminent. Uncontrolled aircraft will be alerted upon entry into a Terminal Control Area (TCA). Obstacles and restricted airspace regions will also be stored for access by ATARS. A Controller Alert Message is also generated when these kinds of alerts are required.

2.3.6 Altitude Echo

ATARS lets the pilot verify his mode C reported altitude and his manually entered correction to his barometric altimeter reading. This message is sent upon the aircraft first entering ATARS service, periodically thereafter, and also whenever the pilot requests this service. The message uplinks (echos) the corrected mode C report, and also contains the altimeter correction used by ATARS.

2.4 Multi-site Considerations

ATARS is to be implemented in a complete system by performing the ATARS function in the same digital computer facility that is resident at each DABS sensor site. Hence, ATARS is implemented as a distributed function and must be provided with a means for coordination between adjacent ATARS functions.

The necessary coordination between ATARS functions can be achieved by providing direct ground communication links between adjacent DABS sites or by using the information stored in the Resolution Advisory Register (RAR). This register is on board each aircraft which is equipped to receive ATARS service. Each ATARS function performs ATARS calculations for all aircraft within a specified geographical area which represents the area of responsibility of that ATARS. These areas of responsibility overlap in the vicinity of their boundaries to form seam areas in which two or more ATARS functions may have responsibility. The generation of incompatible resolution advisories to a pair of aircraft by two different ATARS functions is prevented by assigning a priority ordering to sites which provide service in the seam between sites. The site which sees both the aircraft and has the highest priority is allowed to resolve the conflict.

2.5 ATARS - BCAS Coordination

The coordination of ATARS and BCAS is through the RAR. The RAR is a resolution advisory storage device on board each BCAS and ATARS equipped aircraft. This device is read by examining replies to BCAS and DABS sensor interrogations. The current resolution advisories generated by either BCAS or ATARS are taken as constraints when either system selects maneuvers for a new conflict.

3. HIGH LEVEL PROCESSING AND SYSTEM DATA STRUCTURES

3.1 High Level Processing

This section discusses the execution control, the sequencing of tasks, and the external interface of ATARS sector processing.

3.1.1 Sector Processing

Special consideration must be given to the proper interplay and overall control of the sector oriented task sequencing of ATARS. Executive control must arrange for smooth transitions of control and effective utilization of computer time resources. Although a precise implementation of an executive program is not specifically addressed, a solution is outlined in the diagram of Figure 3-1.

Sector processing in ATARS provides a method to take small, defined areas (sectors) of the ATARS surveillance area and process the data from each sector as a group. The ATARS sectors illustrated in Figure 3-1 contain two antenna sectors of 11 1/4 degrees each. (The term "sector" used unqualified should be interpreted as an ATARS sector (22 1/2 degrees) while reference to an antenna sector (11 1/4 degrees) will always be written as "antenna sector".) Because of a generalized design approach for sector processing, the requirement for an ATARS sector to contain two antenna sectors is flexible and may be site adaptable. Care must be taken when enlarging the ATARS sectors in that a larger area would contain a larger data base and each sector's processing time would have to be adjusted. Also, certain sector and time dependent parameters need to be adjusted.

The report-to-track correlations provided by the DABS sensor are accepted by ATARS as they are received because ATARS is an uncorrelating user. Track data which is tranferred from DABS to ATARS is slaved to the antenna rotation. Target reports arrive in the buffer area in a batch (one antenna sector's worth of reports) once per 11 1/4 degree antenna sector. The DABS sensor triggers the ATARS executive to read the report buffer which includes a header containing the sector identification and sector time.

The real-time processing rate of ATARS is maintained in synchronization with the DABS sensor beam. This antenna sector synchronization is important because the executive program must order tasks to be initiated and terminated for the sector's data at discrete times in the processing scan. These times, noted as critical times in Figure 3-1, refer to the start of a particular sector in the processing scan (e.g., Note critical point 3.

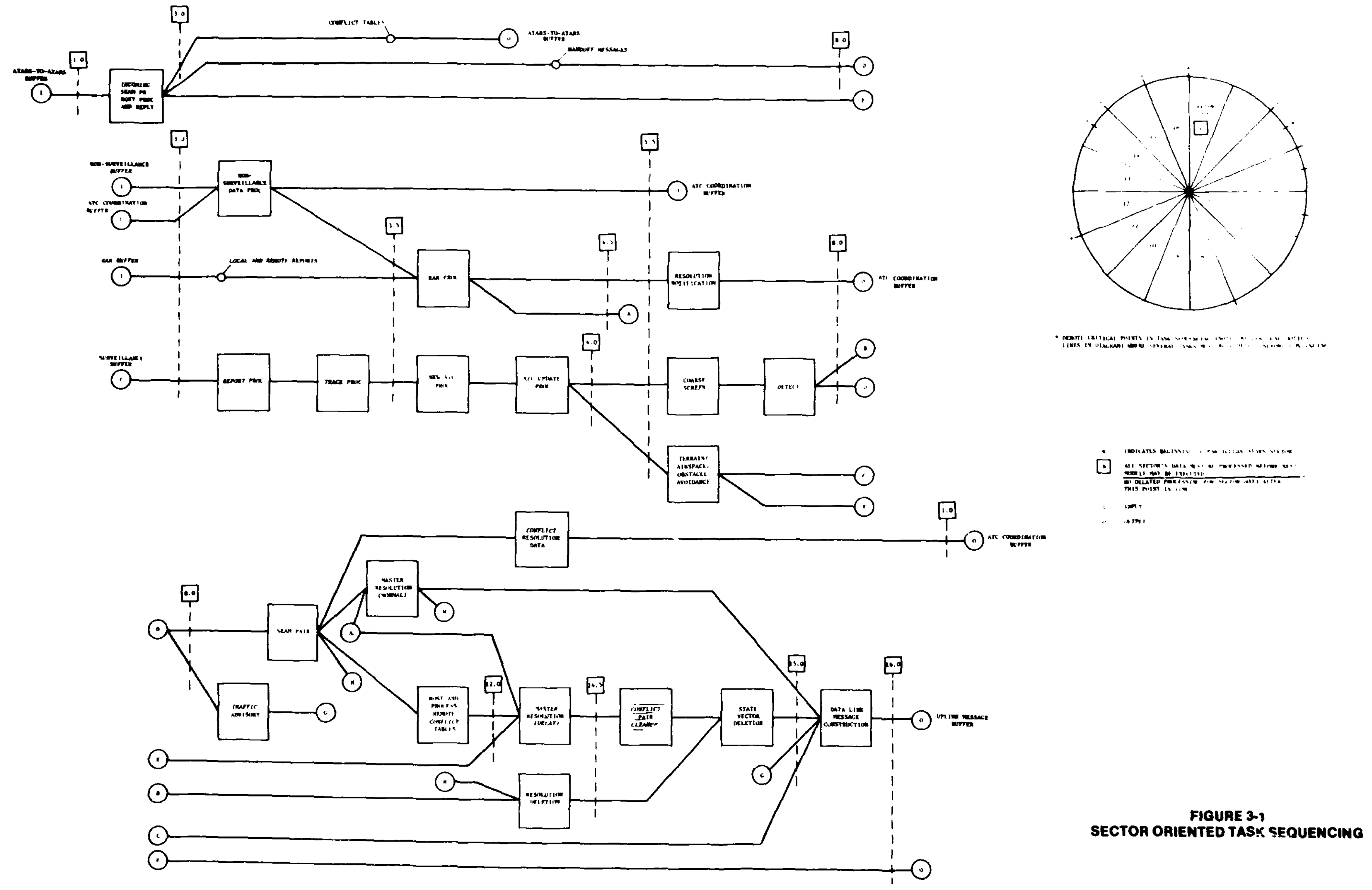


FIGURE 3-1
SECTOR ORIENTED TASK SEQUENCING

This means no data for sector 1 is available for the Report Processing Task before the start of sector 3). The sector numbers in the diagram illustrate the sequencing of tasks for the track data in sector 1, but are easily related to any sector by just adding the desired sector's identification number minus one to the critical point (e.g., Critical point 3 on the diagram is the start of sector 3 when referencing the processing of sector 1 data. If sector 4 data is the reference, then critical point 3 on the diagram becomes 3 + (4-1), or sector 6).

The sector processing diagram illustrates all the tasks that must be executed within one scan for each sector of data. The task sequencing and data flow are shown by the solid connecting lines. Each step in the process is dependent on one or more tasks being completed or an input buffer being filled. When two or more tasks must be completed for a new task to start, a critical point in sector processing is noted. A summary of the time span required or allowed for each task in a single scan is outlined in Table 3-1. The executive program controls the initiation and termination of each task according to the window length for each task. With the starting times and processing windows allowed for the various tasks, several tasks throughout the processing cycle may run in parallel while processing a particular sector. The executive determines that each sector is processed in a step-by-step manner throughout the ATARS process. At the same time, the executive program controls and determines when each task is ready to accept the next sector for processing as critical points are reached in the task sequencing. The executive program handles the major data structures (Table 3-2) for the tasks by providing pointers to each sector of data in the data structure and by placing data in the various structures. This keeps the data segregated according to sectors where required. Care must be taken by the executive to make sure that data structures and lists for a particular sector are not being updated and read at the same time. A mechanism for lockouts must be implemented to prevent this possibility.

One delay is required during the task sequencing and must be implemented by the executive. This delay is required to make sure that up-to-date aircraft positions and velocities are used when determining potential conflicts or resolving old conflicts with the sector being processed. This delay occurs after execution of the Aircraft Update Processing Task and the new position in the data base has been established for the aircraft in the sector. (The aircraft are ordered in the data base according to their x coordinate in order to expedite processing in succeeding tasks.) In order to have current positions for aircraft in the two adjacent sectors, further processing of the current sector is delayed until aircraft in the next two sectors have been updated.

TABLE 3-1

TIMING WINDOWS FOR ATARS FUNCTIONAL PROCESSES

ATARS FUNCTION	WINDOW START (Processing Length-Sectors)
1. Downlink	N (2.0)
2. Incoming Seam Pair Request Processing and Reply (Conflict Tables)	N (2.0)
3. Surveillance Report Processing	N+2 (0.5)
4. Non-surveillance Message Processing	N+2 (0.5)
5. Track Processing	N+2 (0.5)
6. RAR Processing	N+2.5 (1.0)
7. New Aircraft Processing	N+2.5 (0.5)
8. Aircraft Update Processing	N+2.5 (0.5)
9. Resolution Notification	N+4.5 (1.5)
10. Terrain/Airspace/ Obstacle Avoidance	N+4.5 (9.5)
11. Coarse Screen	N+4.5 (2.5)
12. Detect	N+4.5 (2.5)

TABLE 3-1
(Concluded)

ATARS FUNCTION	WINDOW START (Processing Length-Sectors)
13. Traffic Advisory	N+7 (7.0)
14. Seam Pair	N+7 (4.0)
15. Master Resolution (Normal)	N+7 (6.5)
16. Request and Process Remote Conflict Tables	N+7 (4.0)
17. Conflict Resolution Data	N+7 (9.0)
18. Resolution Deletion	N+7 (6.5)
19. Master Resolution (Delayed)	N+11 (2.5)
20. Conflict Pair Cleanup	N+13.5 (0.5)
21. State Vector Deletion	N+13 (0.5)
22. Data Link Message Construction	N+14 (1.0)
23. Uplink	N+15 (1.0)

TABLE 3-2

MAJOR DATA STRUCTURES FOR ATARS SECTOR PROCESSING

DESCRIPTION (Sector Requirement)	REFERENCE
Surveillance Input Data (Antenna Sector)	3.2
State Vector (None)	3.3.1
PWILST (None)	3.3.2
Conflict Table and Pair Record (None)	3.3.3
Encounter List (ATARS Sector)	3.3.4
XINIT List (ATARS Sector)	4.5.1.3
X-list and EX-list (ATARS Sector Threading)	6.1.1
ATARS Sector List (ATARS Sector)	6.1.2
Potential Pair List (ATARS Sector)	7.4
Resolution Pair Acknowledgement List (None)	11.2
Controller Alert List (None)	11.2
Deletion List (ATARS Sector)	15.3

3.1.2 Task Timing and Sequencing

Figure 3-1 presents the highest level flow diagram, which displays the sequencing of all the major tasks for ATARS sector processing. The major delivery points for the input/output buffers are indicated on the diagram. The numbers in the boxes denote critical points in the task sequencing where several tasks must be completed before starting the next task. It is required that the tasks be executed in the order shown in Figure 3-1 for each sector of data. Each task in the sector processing sequence has a defined "window" in which all computations for the particular sector of data must be completed (see Table 3-1). The individual tasks involve other routines and processes which are necessary to complete the assignments for a given task.

The following discussion describes the operation of the ATARS sector processing at the highest level and represents the performance of all tasks on data from one ATARS sector. Through the executive control, this sector process is applied individually to the data from all ATARS sectors in the manner described in Section 3.1.1.

The first major input data processor is the Non-surveillance Message Processing Task which accepts non-surveillance data from DABS on an antenna sector basis through the Non-surveillance Buffer. The messages are processed once per antenna sector at the initiation of report processing. The output of the non-surveillance task is used to update the aircraft state vectors accordingly.

The second major input data processor is the RAR Processing Task. RAR information is received through the RAR Buffer. The messages are processed once per sector. The primary function of the RAR processor is to examine the contents of the RAR of aircraft with ATARS equipage each time this data is downlinked and to update the information in the ATARS Conflict Tables accordingly. RAR processing notes the acceptance of resolution advisories uplinked by the local ATARS site and records the existence of resolution advisories generated by other systems. For conflicts involving a controlled aircraft, the RAR Processing Task also updates the Resolution Pair Acknowledgement List to show the controller resolution advisories which have actually been delivered by the various collision avoidance systems.

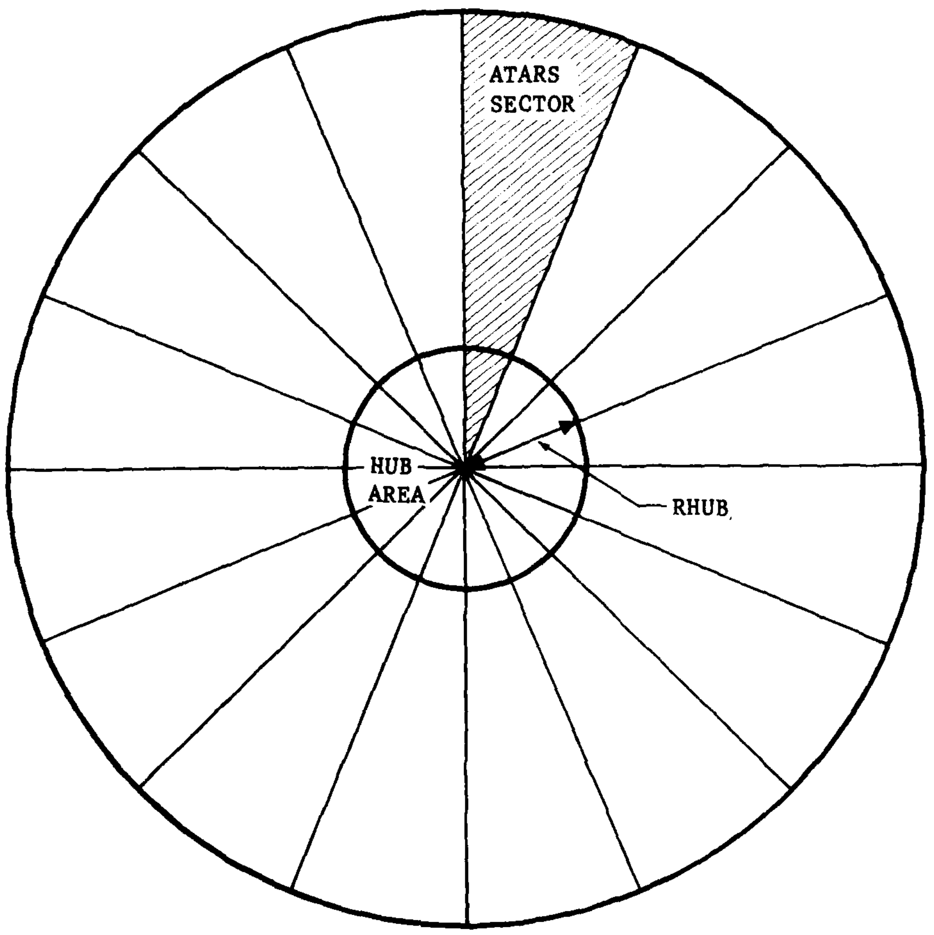
The third major input data processor is the Report Processing Task. The information for this task is received through the Surveillance Buffer and the reports are processed once per antenna sector. A decision is made on whether the reports fall inside the ATARS/Domino surveillance area and, if so, DABS and ATCRBS tracks are initiated with an aircraft state vector and added to the antenna sector list.

A fourth task which processes input data is the Incoming Seam Pair Request Processing and Reply Task. The data for this task is received through the ATARS-to-ATARS Buffer. This buffer operates as a two-way exchange, providing input and accepting outputs from this task. The incoming seam pair request task performs processing and forms replies to messages received over ground lines (during the antenna sweep through sector 1) from neighboring ATARS sites. Each message is a request for Conflict Tables involving a specified pair of aircraft. This task identifies the aircraft in the request and returns own-site's copy of the Conflict Tables, if any.

After report processing has initiated new tracks for the XINIT List or associated reports with existing tracks, the Track Processing Task performs track updates through the smoothing and prediction algorithms. As new tracks are qualified for ATARS service, they are added to a track initiation list for additions to the ATARS data base. These aircraft are then added to the X-list or EX-list in the New Aircraft Processing Task. The new aircraft are linked into the data base to be included with aircraft in the ATARS sector for which they are identified. The Track Processing Task performs additional elimination of tracks which are not to be serviced by ATARS.

Next, each aircraft in the sector has its position updated to a common sector time in the Aircraft Update Processing Task. This is necessary because track reports are received from both the local sensor and remote sensors, and the data received on all aircraft will have been measured at different times.

The area near the radar site providing the surveillance data for ATARS must be given special consideration in this task during sector processing. This area is designated the hub area and is defined by a circle of radius RHUB (approximately 10 nmi) from the radar site (Figure 3-2). The position of all aircraft in the area must be updated every quarter scan (approximately 1.2 seconds). This is necessary in sector processing because data is processed in sector groups and a small position change in this area may move an aircraft one or more sectors from its last sector location. Thus, an updated sector identification and position is maintained for aircraft in the immediate vicinity of the radar site four times per antenna scan. Using the updated data base, a sector of aircraft is processed in a parallel mode by the following two tasks: Coarse Screen Task and Terrain/Airspace/Obstacle Avoidance Task.



RHUB - RADIUS OF HUB PROCESSING AREA

FIGURE 3-2
DIAGRAM OF HUB PROCESSING AREA AS CONTAINED IN THE ATARS SERVICE AREA

The Coarse Screen Task searches the data base for aircraft that are potentially in conflict with each entry on this sector's list of aircraft. This search is implemented through the use of two independent, doubly linked lists which are ordered on the x coordinates of the aircraft. Each aircraft is contained on one list or the other. Two separate lists are maintained in order to make the search for potential conflict pairs more efficient. All aircraft which would require large search limits because of high speeds or other factors are placed on one list called the EX-list and all other aircraft are placed on the other list called the X-list. In coarse screening each aircraft is compared with its neighbors on its own list (and possibly with some aircraft on the other list, as well) to find all pairs of potentially conflicting aircraft. A pattern of searches has been devised that avoids duplicate detection of pairs. The pairs of aircraft which are identified are entered on the Potential Pair List for this sector.

The Terrain/Airspace/Obstacle Avoidance Task has the capability to provide an alert for the violation of restricted airspace, close proximity to the terrain, and close proximity to an obstacle. This task operates on the sector list of aircraft and only affords service for those aircraft in the ATARS service area. The logic to determine the need for an alert is provided in this task, while the actual construction of the uplink message is performed by the Data Link Message Construction Task later in the sector processing sequence.

Processed in parallel with the above two tasks is the Resolution Notification Task. This task processes all pairs on the Resolution Pair Acknowledgement List for the sector. The generation of Resolution Notification Messages to the controller occurs in the Resolution Notification Task.

The Potential Pair List for the sector is used by the Detect Task. Detection determines if ATARS controller alert or traffic/resolution advisories are required for each pair on the list. The output of the Detect Task is an Encounter List entry which indicates if a controller alert, proximity advisory, threat advisory or resolution advisory are required.

If the pair was previously in resolution status and no longer requires resolution, it is flagged for resolution deletion. Pairs on the Encounter List are input to the Traffic Advisory and Seam Pair Tasks which determine the correct action for the pair. The Traffic Advisory Task creates, updates, and deletes entries on a list maintained for each subject aircraft. Entries on the list contain data for other aircraft which are in conflict with the subject aircraft. This data is used later to generate traffic advisory messages. Pairs which require resolution are processed by the Master Resolution Task, either Normal or Delayed.

Those pairs on the Encounter List which are flagged for resolution deletion are processed by the Resolution Deletion Task. This task has the general purpose of ensuring that conflict pair data in the Conflict Tables is closed out in the proper manner when it is no longer needed. This task initiates the uplink of null resolution advisories for conflicts which were resolved by the local site and deletes data for any aircraft flying out of the local site's coverage area.

The Encounter List is used as input for the Seam Pair Task. This task determines own-site resolution responsibility for each pair on the Encounter List. If own-site is responsible and either aircraft is in an ATARS seam, the task flags the pair for delayed resolution. Normal resolution is allowed if neither aircraft is in a seam. The controller alert status is also set or cleared according to own-site's resolution responsibility.

The Conflict Resolution Data Task is available for processing as soon as there are entries on the Encounter List requiring a controller alert. The task creates and updates from the information on the Encounter List entries on the Controller Alert List. When three of the last five scans have had a controller alert flag set in the Detect Task, or have had the immediate controller alert flag set in detect, a Controller Alert Message is generated containing conflict resolution data.

The Master Resolution (Normal) Task provides the framework for the initial selection of resolution advisories, the monitoring of the conflict to adjust resolution advisories to more restrictive or less restrictive maneuvers as the situation warrants, the staging of advisories in an uncontrolled/controlled encounter, and the recomption of resolution advisories when the initial maneuvers are ineffective or incompatible with advisories from another source. This is accomplished through the use of the Encounter List, Pair Records, and Conflict Tables. The logic for providing the selection of the best resolution maneuver for a pair of aircraft given the current set of constraints is the Resolution Advisories Evaluation Routine which is called by the Master Resolution Task. This logic performs a fast-time simulation of all possible sets of maneuvers and selects the one that will provide the greatest safety after considering many factors. Some of these factors are the separation at closest approach, the turn status of each aircraft, the likelihood of a domino conflict, and the vertical and horizontal maneuver performance of the aircraft. The logic evaluates multi-aircraft situations by considering the current maneuver constraints when determining resolution advisories for a new conflict. If the Resolution Advisory Evaluation Routine is unable to select resolution advisories for a conflict pair when called by the Master Resolution (Normal) Task, then the pair is flagged for processing by the Master Resolution (Delayed) Task.

Before the completion of the Master Resolution (Normal) Task, the Master Resolution (Delayed) Task is initiated for the sector's aircraft seam pairs and pairs not resolved by the Master Resolution (Normal) Task. The Master Resolution (Delayed) Task provides the same service to the aircraft pairs as the Master Resolution (Normal) Task.

At the completion of normal and delayed master resolution, the Conflict Pair Cleanup Task is initiated. The Conflict Pair Cleanup Task serves as a backup to the Resolution Deletion Task to ensure that data in the Conflict Tables is closed out in the proper manner when it is no longer needed. Primary input to the task is the linked list of Conflict Tables.

After the completion of the Conflict Pair Cleanup Task, the State Vector Deletion Task is initiated for the sector. This task deletes the State Vector and ends tracking of an aircraft which leaves the ATARS/Domino surveillance area. If the aircraft is involved in a conflict, an entry is made on the remote list of aircraft. If ATARS has unfinished business with the aircraft, the above actions are inhibited.

The last task to be executed in the sector seque ing is the Data Link Message Construction Task. This task processes the sector's list of aircraft and generates all messages required to be uplinked to each aircraft. The task reads the PWILST and Conflict Tables and generates traffic advisories and resolution advisories for aircraft in conflict. If the aircraft is in restricted airspace, or in proximity to the ground or other obstacles, it generates a message containing warning data.

The Data Extraction Function records important ATARS parameters, suitable for reconstructing a particular aircraft pair's progress through the ATARS system. This information can be used for analysis, system error tracing, or as an event log. The logic for this task does not exist as a separate entity, but is interspersed among all ATARS software. The code utilized and consequently the information extracted is a system option.

The Status Monitoring and Reporting Function monitors all of the

ATARS tasks and major files for failure or marginal operation conditions. It determines whether ATARS can be classified as operating normally, degraded, or has failed. It reports the ATARS status to DABS and to ATC. It can provide, upon ATC request, a complete list of degraded or failed conditions it has detected.

3.2 ATARS Interfaces

The ATARS interfaces go through the DABS sensor. Even though ATARS sends messages to ATC facilities, aircraft and other sites, and receives messages from these sources, all communications are handled through the local DABS sensor. The exact physical character of the interfaces and the buffer formats between DABS and ATARS are found in Reference 1. The contents of the surveillance report formats are discussed in this section for clarification.

A diagram of the ATARS-SENSOR interface is illustrated in Figure 3-3. Note that some of the information flows in one direction only (DABS-to-ATARS, ATARS-to-DABS) and the ATARS-to-ATARS information flows in a two-way buffer. The ATARS buffers noted in the diagram are serviced by the appropriate task in the ATARS software.

The input-only information from the DABS sensor to ATARS consists of reports in the following four buffers:

1. RAR Buffer

2. Surveillance Buffer

3. Non-surveillance Buffer

4. ATC Coordination Buffer

These buffers are written by the DABS sensor and read by ATARS. They are two-segment buffers with one segment being written at the same time that the other segment is read. Overwriting a segment which is being read must be prevented by a lockout flag or by careful program timing.

Input data for the Surveillance Buffer is transmitted in blocks consisting of all the reports available to the local sensor (local or remote reports) during one 11 1/4 degree antenna sector of local antenna rotation. A single completion interrupt is then given to the ATARS processor. Just before the interrupt, a special header word is filled in each Surveillance Buffer with the current local antenna sector identification number and antenna sector time (see Figure 3-4).

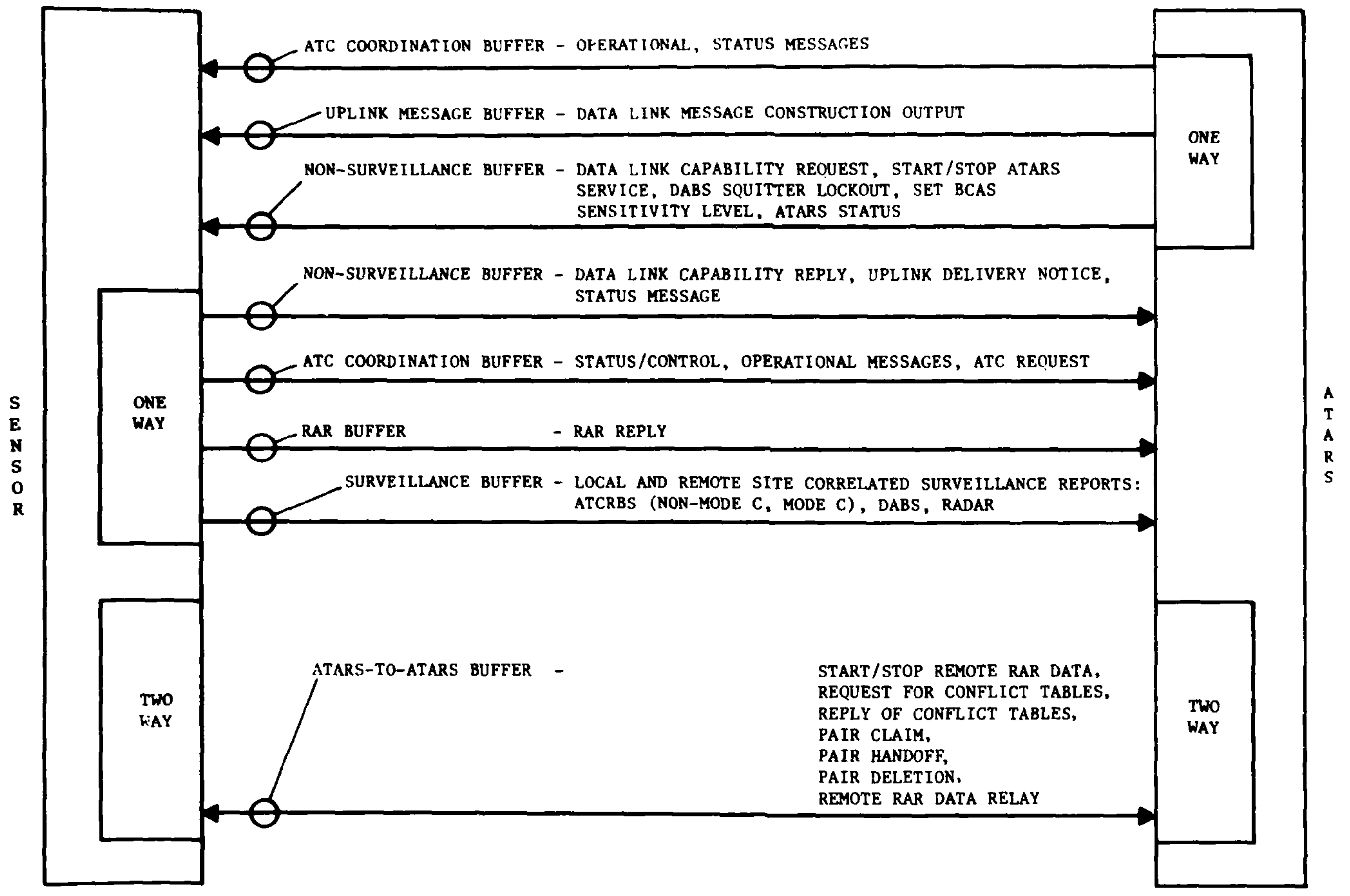

ATC COORDINATION BUFFER - OPERATIONAL, STATUS MESSAGES
UPLINK MESSAGE BUFFER - DATA LINK MESSAGE CONSTRUCTION OUTPUT
NON-SURVEILLANCE BUFFER - DATA LINK CAPABILITY REQUEST, START/STOP ATARS SERVICE, DABS SQUITTER LOCKOUT, SET BCAS SENSITIVITY LEVEL, ATARS STATUS
NON-SURVEILLANCE BUFFER - DATA LINK CAPABILITY REPLY, UPLINK DELIVERY NOTICE, STATUS MESSAGE
ATC COORDINATION BUFFER - STATUS/CONTROL, OPERATIONAL MESSAGES, ATC REQUEST
RAR BUFFER - RAR REPLY
SURVEILLANCE BUFFER - LOCAL AND REMOTE SITE CORRELATED SURVEILLANCE REPORTS: ATCRBS (NON-MODE C, MODE C), DABS, RADAR
ATARS-TO-ATARS BUFFER - START/STOP REMOTE RAR DATA, REQUEST FOR CONFLICT TABLES, REPLY OF CONFLICT TABLES, PAIR CLAIM, PAIR HANDOFF, PAIR DELETION, REMOTE RAR DATA RELAY
SENSOR
ONE WAY
TWO WAY
ATARS
ONE WAY
TWO WAY

FIGURE 3-3
ATARS-SENSOR INTERFACE DIAGRAM

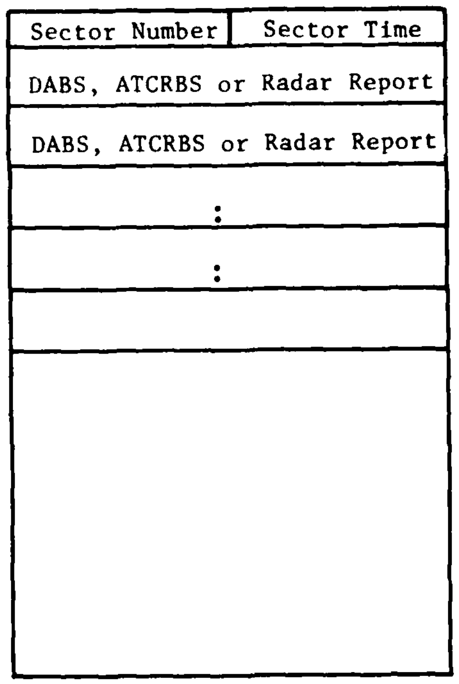


FIGURE 3-4
SURVEILLANCE BUFFER DATA STRUCTURE

The Surveillance Buffer contains reports utilized by the ATARS Report Processing Task. The local or remote reports are disseminated to ATARS using the formats given in Table 3-3 for DABS, Table 3-4 for ATCRBS reports and Table 3-5 for radar-only reports. These reports are outputs of the DABS sensor tracker correlator; uncorrelated reports do not occur. These reports are accepted in their entirety, as ATARS is a non-correlating user of DABS.

The RAR Buffer contains all the RAR information from the RAR equipped aircraft. The RAR Buffer contents are continuously read by the RAR Processing Task and the information is passed along to the appropriate ATARS tasks.

The Non-surveillance Buffer contains other messages not related to surveillance processing. The buffer is read on a continuing basis by the Non-surveillance Message Processing Task. The formats of these messages are indicated in Section 5.

The ATC Coordination Buffer contains all the messages being passed from ATC to ATARS. The sensor acts merely as a medium to transfer the messages on a timely basis. These messages are read on a continuing basis by ATARS.

The output-only information from ATARS to the DABS sensor consists of reports in the following four buffers:

1. Uplink Message Buffer

2. Non-surveillance Buffer

3. ATC Coordination Buffer

4. Data Extraction Buffer

The ATARS messages generated for the aircraft through the Data Link Message Construction Task are delivered to the Uplink Message Buffer. All non-surveillance messages generated by ATARS are delivered to the Non-surveillance Buffer. ATC message and Data Extraction information are transferred to the appropriate buffer.

The ATARS-to-ATARS buffer provides information from local ATARS to remote ATARS and remote ATARS information for local ATARS. This buffer contains the communication necessary between ATARS sites that need not be acted on by the sensor.

TABLE 3-3

DABS REPORT FORMAT[1]

FIELD	LENGTH (Bits)
Test	1
Format Type	2
Radar Substitution	1
Mode C Present	1
Sensor Priority Status	1
SPI (IDENT)	1
Radar Reinforced	1
Code 7700	1
Code 7600	1
Alert	1
Target Control State	1
Reply Type	3
Null Report	1
Track Start	1
Track Drop	1
Range[2], LSB = 1 Ru (1/16 us)	16
Azimuth, LSB = 1 Au (0.022°)	14
Mode C Altitude, LSB = 100 feet	12
DABS ID	24
Sensor ID	4
Time of Day	24
Diffraction Flag	1
Altitude Correction	8
Zenith Cone Flag	1
ATARS UM Field	6
Non-empty RAR Condition	1
Spares	8

[1]The exact form of this data is provided in Reference 1.
[2]This is a two-way range expressed in units of time.

TABLE 3-4

ATCRBS REPORT FORMAT[1]

FIELD	LENGTH (Bits)
Test	1
Format Type	2
Radar Substitution	1
Mode 3/A Present	1
Mode C Present	1
Mode C Not Decoded	1
SPI (IDENT)	1
Radar Reinforced	1
Code 7700	1
Code 7600	1
Target Control State	1
False Target Flag	1
Correlation Confidence	1
Null Report	1
Track Start	1
Track Drop	1
Range[2] LSB = 1 Ru (1/16 us)	16
Azimuth, LSB = 1 Au (0.022^{o})	14
Mode C Altitude, LSB = 100 feet	12
Mode 3/A	12
ATCRBS Surveillance File No.	12
ATCRBS Code in Transition	1
Sensor ID	4
Time of Day	24
Diffraction Flag	1
Altitude Correction	8
Zenith Cone Flag	1
Spares	8

[1]The exact form of this data is provided in Reference 1.
[2]This is a two-way range expressed in units of time.

TABLE 3-5

RADAR REPORT FORMAT[1]

FIELD	LENGTH (Bits)
Test	1
Format Type	2
Null Report	1
Track Start	1
Track Drop	1
Range[2], LSB = 1 Ru (1/16 us)	16
Azimuth, LSB = 1 Au (0.022°)	14
Radar Surveillance File No.	12
Sensor ID	4
Time of Day	24
Spares	8

[1]The exact form of this data is provided in Reference 1.
[2]This is a two-way range expressed in units of time.

3.3 System Data Structures

There are four main data structures used throughout the ATARS algorithms. They are introduced in this section and discussed in detail in the sections referenced below.

3.3.1 Aircraft State Vector

The aircraft State Vector used by the ATARS processor is presented in pseudocode in the System Data Structures. The State Vector contains all the known information for a particular aircraft that is being tracked by ATARS. The data is listed in the pseudocode under several distinct categories:

Pointers

Horizontal Tracker Data

Vertical Tracker Data

Times

Flags

General Values

Domino Projections

Data Block

The nomenclature for each data variable in this document may be used with the number 1 or 2 added as a suffix (e.g., XDE, XDE1, XDE2). When the variables appear without the numerical suffix, a single aircraft is being addressed in the discussion. In such a single aircraft situation, variables may also appear with a 1 as a suffix. When two aircraft are being compared, it is necessary to identify the variables from each aircraft's State Vector. To do this, a 1 is added to the subject aircraft's State Vector and a 2 is added to the object aircraft's State Vector variables.

The individual aircraft State Vectors are placed in a file called the Central Track Store (CTS). The CTS is a convenient location to access all tracks for the ATARS processor. Detailed information for the CTS usage is found in Section 4.3.

3.3.2 PWILST

The PWILST is a data structure, accessed from an aircraft State Vector, which stores traffic and other non-resolution advisories. Every entry has a TYPE and several data fields. The entries are created by the Traffic Advisory Task and by several other tasks. A detailed explanation is given in Section 9.2. This list is maintained from scan to scan as long as ATARS service continues.

3.3.3 Conflict Table and Pair Record

This table is a data structure, accessed from the aircraft State Vector, for each aircraft involved in a conflict. Each Conflict Table contains information on all the aircraft in the cluster, linkage to other Conflict Tables, a pointer to the list of Pair Records, and Conflict Table entries. The Conflict Table entries contain the information about the aircraft in relation to the conflict cluster, resolution advisories, and displayed advisories. Conflict Tables contain a Pair Record for every aircraft pair declared in conflict. The Pair Record contains information that is unique to a particular pair in the Conflict Table. The Conflict Table entry is created by the Seam Pair Task and is maintained by the Master Resolution Task and by several other tasks. A detailed explanation is given in Section 12.2. This data is maintained from scan to scan as long as the conflict remains and is deleted by the Resolution Deletion Task or the Conflict Pair Cleanup Task.

3.3.4 Encounter List

This list is created by the Detect Task as it processes each sector's Potential Pair List. A separate Encounter List is kept for each sector's data. The entries on this list indicate any further processing required for the pair: controller alert, traffic advisory, resolution advisory (including provisional responsibility, or delayed resolution), BCAS inhibit for ATCRBS threat, or resolution deletion. Also, certain calculations made and flags set for the pair are saved to be used by later tasks. These tasks may alter the status flags as they process a pair's entry. The list is discarded at the end of each scan.

3.4 En Route Operation

ATARS is required to provide service in en route areas as well as terminal areas. Certain characteristics of the en route environment require that the ATARS system operating in the en route area differ slightly from that in the terminal area. The body of this document addresses the ATARS system in a terminal area. This section describes the ways in which ATARS in the en route area differs.

3.4.1 ATARS Operation With Back-to-Back DABS Antenna

DABS sensors are to be installed at sites where current ATC en route radars are in operation and are to be operated in conjunction with the en route primary radars. The primary radars operate with a scan time of 10 to 12 seconds. If the DABS sensor were required to operate with a data rate corresponding to this scan time, the ATARS service that could be provided would be unacceptable. To improve the ATARS service, the DABS sensor has been designed to operate with a back-to-back antenna (an antenna with two faces directed 180 degrees apart) rotating with a scan time of 10 to 12 seconds. The effective data update interval is then five to six seconds.

The operation of the DABS sensor with the back-to-back antenna is described in Reference 1. The modifications to the normal ATARS algorithms that are required for operation with the back-to-back antenna are described in this section. The remainder of this document describes algorithms for operation with a normal DABS sensor (scan time on the order of 4.7 seconds).

An antenna sector will still be defined as a sweep of 11 1/4 degrees. When an input completion interrupt is received, data for two antenna sectors will be passed through the Surveillance Buffer. This data will be the data collected from the front face and the back face of the antenna while the antenna rotated through 11 1/4 degrees. The two antenna sectors represented are 180 degrees apart. The data from both antenna sectors will be serviced by the report processing algorithms. Sector header data will be provided for each antenna sector of data.

The ATARS sector processing executive logic must be modified so that the ATARS Sector List of aircraft from each face of the antenna can be processed as a separate stream. When a new antenna sector is entered, the IDs of both the front-face and the back-face antenna sectors will be added to the antenna sector request stack. The sectors will then be processed as separate streams through the ATARS logic.

The time allowed to complete each processing task must be adjusted so that advisories detected on the front-face of the antenna will be available for uplink on the next sweep of the back-face of the antenna. The primary radar associated with an en route DABS sensor will have only one antenna face. This face will be matched with the front-face of the DABS sensor antenna. Hence, radar-only tracks will be processed once per scan. It is necessary to have antenna position reports supplied to ATARS once per antenna sector for only the front-face.

In prediction, with the back-to-back antenna, positions should be predicted to the time of next expected data. This will be a prediction over a half scan rather than a full scan. ATARS will be able to deliver resolution advisories on the uplink on either the front-face beam or the back-face beam.

3.4.2 Modification of ATARS Detection Parameters

Another characteristic of the en route area, in addition to the slower scan rate, is the operation with larger aircraft-to-sensor ranges and a resultant reduction in position and velocity tracking accuracy. To provide acceptable operation at larger ranges, ATARS must use increased conflict detection parameters. Currently, ATARS tests the aircraft in a pair before selecting a set of detection parameters for that pair. If either aircraft is outside a specified area, the detection parameters for that pair are increased. Additional parameters that are specifically related to the en route environment will be supplied in the future.

3.5 Pseudocode for System Data Structures

The following section contains the global parameter and variable structures for ATARS. In addition to system structures, groups of structures for specific tasks are also included.

For all BIT parameters and variables, the comment gives the TRUE meaning of the value.

PSEUDOCODE TABLE OF CONTENTS

```
--------------------------------------------------------------------------------

                         <*** COARSE SCREEN PARAMETERS  ***>

STRUCTURE CSCREEN


  GROUP thresholds

     FLT AHI              < altitude threshold >

     FLT VPCS             < vertical proximity test limit >

     FLT XSP              < spacing between signposts on X/EX_lists >

     FLT ZFAST            < z velocity threshold >


  GROUP distances

     FLT RPVI             < maximum range for proximity advisory >

     FLT RMAXE            < maximum distance traveled by object aircraft on
                             EX-list plus protection envelope >

     FLT RMAXI            < maximum distance traveled by object controlled
                             aircraft on X-list plus protection envelope >

     FLT RMAXV            < maximum distance traveled by object uncontrolled
                             aircraft on X-list plus protection envelope >


  GROUP times

     FLT TLI              < look ahead time for controlled AC >

     FLT TLV              < look ahead time for uncontrolled AC >


ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
---------------------------------------------------------------------------------

      <*** AREA/ZONE PARAMETERS (see Tables 8-6 and 8-7 for explanation) ***>
STRUCTURE AZPARM

  GROUP coarse_region
     FLT ZMMNX              < defines lower x bound of coarse screen
                               region for area determination >
     FLT ZMMXX              < defines upper x bound of coarse screen
                               region for area determination >
     FLT ZMMNY              < defines lower y bound of coarse screen
                               region for area determination >
     FLT ZMMXY              < defines upper y bound of coarse screen
                               region for area determination >
     FLT ZJMNX              < defines lower x bound of coarse screen
                               region for zone determination >
     FLT ZJMXX              < defines upper x bound of coarse screen
                               region for zone determination >
     FLT ZJMNY              < defines lower y bound of coarse screen
                               region for zone determination >
     FLT ZJMXY              < defines upper y bound of coarse screen
                               region for zone determination >

  GROUP counts
     INT NOI                < number of Type 1 areas >
     INT NOII               < number of Type 2 areas >
     INT NOZ1               < number of Type 1 zones >
     INT NOZ2               < number of Type 2 zones >

  GROUP arznvb
     INT COAA2              < parallel to final approach zone threshold>
     INT ROTST              < distance from radar where area 4 begins>
     INT ZZON2              < depth of zone 2 glide slope >

ENDSTRUCTURE;


---------------------------  ATARS GLOBAL PARAMETERS  ----------------------------
```

```
-----------------------------------------------------------------------------------------

                    <*** DOMINO AND DETECT PARAMETERS ***>

STRUCTURE DETPARM


  GROUP general_parameters

    FLT ADET                < sets miss dist threshold (DSQ) for modified Tau >

    FLT AFDET               < prefiltering immediate altitude threshold >

                            < maximum of ZAFCON, AFCON, AF, AIFR, AFPWI, AFIFR >

    FLT BDET                < sets miss dist threshold (DSQ) for modified Tau >

    FLT DOTTH               < advisory divergence threshold >

    FLT RDET                < prefiltering immediate range threshold >

                            < maximum of ZRCON2, RCON2, RCMD2, RIFR2, RFPWI2,

                              RFIFR2 >

    FLT VMDTH               < relative horizontal velocity threshold (squared)

                               for computing miss distance squared >

    FLT VRZTH               < variable to prevent division by zero >


ENDSTRUCTURE;


STRUCTURE RAPARM            < Resolution advisory general parameters >


  GROUP filter_thresholds

    FLT DVDPT               < maximum projection time used in VDL logic >

    FLT BZP                 < buffer zone definition expressed as percent of

                               distance >

    FLT BZP2                < BZP**2>


  GROUP times

    FLT TTM                 < number of scans to use in immediate RA logic >

ENDSTRUCTURE;
```

```
---------------------------------------------------------------------------------

STRUCTURE THRSPARM          < Threshold determination parameters >


  GROUP ratios

     FLT VRATC              < significant speed differential >

     FLT VRATTH             < lookahead time determination for equipped/unequipped >

     FLT VRZCON             < slow-close/divergence relative speed threshold >


ENDSTRUCTURE;
```

```
-------------------------------------------------------------------------------

                <*** RESOLUTION ADVISORY SELECTION PARAMETERS ***>

STRUCTURE RRSADV


  GROUP thresholds

     FLT HDTTHSQ            < HDTHSQ threshold when either aircraft turning >

     FLT HDTHSQ             < negative horizontal resolution advisory threshold,
                              neither aircraft turning >

     FLT SEP1               < minimum acceptable 3-dimensional separation >


ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

                       <*** SYSTEM GLOBAL PARAMETERS ***>

STRUCTURE SYSTEM


  GROUP coverage

     PTR CTRTBL(15)          < table of pointers to list of center zone maps >

     PTR MAPTBL(15)          < table of pointers to list of ATARS service maps >

     BIT MASTRTBL(15)        < table of Master Status flags >

     PTR SQMAP               < pointer to squitter lockout map >


  GROUP tracker

     FLT ALO                 < altitude boundary for X/EX_list >

     FLT SPLO2               < square of maximum assumed speed for AC below
                                10,000 ft MSL >

     FLT ZNOM                < altitude used in slant range correction when no
                                altitude measurement is available >


  GROUP miscellaneous

     BIT DOMNONC             < this site providing domino feature for non-mode C >

     FLT HUBRAD              < radius of hub processing area >

     INT OWNID               < ATARS site ID 4-bit field >

     FLT SCANT               < time interval for one radar scan >

     FLT VWEIGHT             < weighting factor for vertical dimension >


  GROUP ztrack

     FLT DT                  < nominal time between updates - one radar scan >

     FLT Q                   < quantization bin width >


ENDSTRUCTURE;
```

```
------------------------------------------------------------------------------------

                         <*** CONFLICT TABLE HEAD ***>

STRUCTURE CTHEAD


  GROUP maintenance

     PTR FCTE                  < first entry in this conflict table >

     PTR NEXTCT                < next conflict table >

     PTR PREVCT                  previous conflict table >


  GROUP data

     INT NAC                   < number of AC in this conflict table >

     PTR PLIST                 < first pair record for AC in this table >

     BIT SEAM                  < seam conflict >


ENDSTRUCTURE;


                        <*** CONFLICT TABLE ENTRY ***>

STRUCTURE CTENTRY


  GROUP maintenance

     PTR NXCTE                 < next conflict table entry >


  GROUP data

     PTR ACID                  < ID of AC in conflict (-> state vector ) >

     PTR ACIDH                 < pair record with horizontal resolution advisory >

     PTR ACIDV                 < pair record with vertical resolution advisory >

     INT HMAN                  < horizontal resolution advisory being sent to AC >

     INT HMAND                 < horizontal resolution advisory being displayed on AC >

     INT MULTH                 < count of conflicts resolved using horizontal dimension >

     INT MULTV                 < count of conflicts resolved using vertical dimension >

     INT NCON                  < AC is included in this number of conflicts >

     BIT REMFLG                < AC remote >

     INT VMAN                  < vertical resolution advisory being sent to AC >

     INT VMAND                 < vertical resolution advisory being displayed on AC >


ENDSTRUCTURE;

----------------------------  ATARS GLOBAL VARIABLES  -----------------------------
```

```
-----------------------------------------------------------------------------------

                              <*** PAIR RECORD ***>

STRUCTURE PREC

  GROUP maintenance
     PTR NXTPR               < next pair record, this table >

  GROUP identifiers
     INT ATSID               < processor (ATARS or BCAS) handling this conflict >
     BIT HDOFF               < site named in ATSID has handed off pair >
     INT SECTID              < ATARS sector for the conflict pair >

  GROUP general_values
     BIT PIFR                < controlled AC gets resolution advisories >
     FLT PMD                 < horizontal miss distance (squared) >
     INT POSCMD              < resolution advisory control variable >
     FLT PVMD                < vertical miss distance >
     BIT PWISW               < coarse screen and detection done >
     FLT TSTART              < time when resolution advisories were selected >

  GROUP ac1
     BITS CMDFL              < field of advisories for domino search >
     INT THMAN               < horizontal resolution advisory to this AC from
                                other non-connected sites >
     INT TVMAN               < vertical resolution advisory to this AC from
                                other non-connected sites >
     PTR INTR                < head of domino list for this AC >
     INT MVT                 < AC turn state at time of resolution advisory
                                selection >
     PTR PAC                 < ID of this AC (-> CTENTRY) >
     INT PHMAN               < horizontal resolution advisory to this AC >
     INT PVMAN               < vertical resolution advisory to this AC >
     BIT SEND                < uplink resolution advisory for this AC >
     INT TRKID               < other AC's track ID >


----------------------------  ATARS GLOBAL VARIABLES  -----------------------------
```

```
------------------------------------------------------------------------------------------

  GROUP ac2
     LIKE PREC.ac1

  GROUP model_validation
     BIT MVDONE              < already done validation >
     FLT MVRAIT              < RA initiation time >
     FLT MVVRZ               < relative ZD (ZD2 - ZD1) >

ENDSTRUCTURE:
```

```
-----------------------------------------------------------------------------

        <*** DOMINO AND DETECT DETERMINATION VARIABLES (TABLE 8-3) ***>
STRUCTURE PDVBL


  GROUP miscellaneous

    FLT ACONTH             < vertical range, for TCONV calculation >

    FLT RCONTH             < horizontal range, for TCONH calculation >

    FLT TWARN              < warning time threshold >


ENDSTRUCTURE;
```

```
                         <*** ENCOUNTER LIST ENTRY ***>
STRUCTURE ELENTRY

  GROUP processing_required
     BIT BOFFREQ            < inhibit BCAS_ATCRBS message required >
     BIT CNAREQ             < controller alert required >
     BIT DELREQ             < delayed resolution required >
     BIT RAPROV             < provisional responsibility for resolution >
     BIT RAREQ              < resolution advisory required >
     BIT RDREQ              < resolution deletion required >
     BIT TAREQ              < traffic advisory required >

  GROUP identifiers
     PTR ACID1              < state vector for AC 1 >
     PTR ACID2              < state vector for AC 2 >
     INT CPSID              < sector ID for the encounter >

  GROUP computed_separations
     FLT ALT                < absolute current altitude separation >
     FLT DOT                < dot product of relative separation and
                              relative velocity vector >
     FLT MD2                < miss distance, squared >
     FLT RANGE2             < horizontal separation, squared >

  GROUP computed_times
     FLT TH                 < time until horizontal sep (DSQ) is violated >
     FLT TV                 < time to coincidence in vertical direction >

  GROUP geographic_dependent
     INT ENAT               < encounter area type >
```

```
--------------------------------------------------------------------------------

 GROUP detect_flags

    BIT CAFLG              < controller alert required, this pair >
    BIT CMDFLG             < resolution advisory required, this pair >
    BIT FPIFLG             < threat advisory required for controlled AC >
    BIT FPUFLG             < threat advisory required for uncontrolled AC >
    BIT ICAFLG             < bypass 3-out-of-5 controller alert logic >
    BIT IFRFLG             < resolution advisory required for controlled AC >
    BIT MTTFLG             < bypass 2-out-of-3 resolution advisory logic >
    BIT PWIFLG             < proximity advisory required, this pair >

ENDSTRUCTURE;
```

```
----------------------------------------------------------------------------------------

                        <*** PWILST ENTRY PROXIMITY TYPE ***>

STRUCTURE TA_PROX


  GROUP maintenance_info

     PTR NXTPWI               < next entry >

     PTR PRVPWI               < previous entry >

     BIT SENT                 < entry has been sent in message >


  GROUP identity

     BIT FND                  < entry not updated this scan >

     PTR OBJ_AC               < state vector entry >

     INT OLD_TYPE             < former type of entry for object AC >

     INT TRACK_NO             < unique number for subject AC TA >


  GROUP rank_data

     INT RANGE_WEIGHTED       < range with vertical component
                                 weighted vs. horizontal>

     INT RANKTYP              < see Table 16-2 for coding >

     INT TAU                  < always zero for this type >


  GROUP advisory_data

     CHR ABBREV               < aircraft abbreviated data >

     BIT ATARS_EQP            < proximity AC equipped with ATARS >

     INT CLIMB_PERF           < climb capability of proximity AC >

     INT CLOCK_BRG            < clock bearing to proximity AC >

     BIT CONTROL              < proximity AC controlled by ATC >

     INT COURSE               < clock heading of proximity AC >

     INT FINE_BRG             < fine increment to clock bearing >

     INT GRND_SPEED           < ground speed of proximity AC >

     FLT RANGE                < slant range, unweighted >

     FLT REL_ALT              < relative altitude of proximity AC >


ENDSTRUCTURE;


------------------------------ ATARS GLOBAL VARIABLES ------------------------------
```

```
--------------------------------------------------------------------------------

                    <*** PWILST ENTRY THREAT TYPE ***>

STRUCTURE TA_THREAT


  GROUP maintenance_info

     PTR NXTPWI                 < next entry >

     PTR PRVPWI                 < previous entry >

     BIT SENT                   < entry has been sent in message >


  GROUP identity

     BIT END                    < entry not updated this scan >

     PTR OBJ_AC                 < state vector entry >

     INT OLD_TYPE               < former type of entry for object AC >

     INT TRACK_NO               < unique number for subject AC TA >


  GROUP rank_data

     INT RANGE_WEIGHTED         < range with vertical component

                                  weighted vs. horizontal>

     INT RANKTYP                < see Table 16-2 for coding >

     INT TAU                    < horizontal tau, pseudo-tau, or large constant >
```

```
---------------------------------------------------------------------------------

 GROUP advisory_data

    CHR ABBREV               < aircraft abbreviated data >

    BIT ATARS_EQP            < proximity AC equipped with ATARS >

    INT CLIMB_PERF           < climb capability of proximity AC >

    INT CLOCK_BRG            < clock bearing to proximity AC >

    BIT CONTROL              < proximity AC controlled by ATC >

    INT COURSE               < clock heading of proximity AC >

    INT FINE_BRG             < fine increment to clock bearing >

    INT FINE_HDG             < fine heading increment to course field >

    INT GRND_SPEED           < ground speed of proximity AC >

    INT HMD                  < predicted horizontal miss distance >

    FLT RANGE                < slant range, unweighted >

    FLT REL_ALT              < relative altitude of proximity AC >

    INT REL_ALT_EXT          < relative altitude extension >

    INT TURN                 < threat strong turn state >

    INT VERT_SPD             < vertical speed of threat >

ENDSTRUCTURE;
```

```
----------------------------------------------------------------------------------------

                    <*** PWILST ENTRY ATCRBS_TRACK_BLOCK TYPE ***>

STRUCTURE ATCRBS_TB


  GROUP maintenance_info

     PTR NXTPWI                < next entry >

     PTR PRVPWI                < previous entry >

     BIT SENT                  < entry has been sent in message >


  GROUP identity

     INT ATCRBS_TRACK_NO       < unique number for subject AC ATCRBS_TB >

     BIT END                   < entry not updated this scan >

     PTR OBJ_AC                < state vector entry >


  GROUP track_data

     INT ALT                   < altitude of ATCRBS AC (ft) >

     INT BEARING               < bearing to ATCRBS AC (deg) >

     INT RANGE                 < range (nmi) >

     INT RANGE_RATE            < range rate (kt) >

     INT VERT_RATE             < altitude rate of ATCRBS (ft/s) >


ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

                     <*** PWILST ENTRY TERRAIN TYPE ***>

STRUCTURE TERRAIN

  GROUP maintenance_info

     PTR NXTPWI              < next entry >

     PTR PRVPWI              < previous entry >

     BIT SENT                < entry has been sent in message >

  GROUP status

     BIT END                 < entry not updated this scan >

     BIT FTAT                < first time advisory transmitted >

  GROUP adv_data

     INT REL_ALT             < altitude of subject AC relative to terrain >

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------------------

                     <*** PWILST ENTRY OBSTACLE TYPE ***>
STRUCTURE OBSTACLE


  GROUP maintenance_info
     PTR NXTPWI              < next entry >
     PTR PRVPWI              < previous entry >
     BIT SENT                < entry has been sent in message >

  GROUP status
     BIT END                 < entry not updated this scan >
     BIT FTAT                < first time advisory transmitted >
     INT OBSTACLE_NO         < identity on stored list of obstacles >

  GROUP adv_data
     INT CLOCK_BRG           < clock bearing to obstacle >
     INT RANGE               < AC range to obstacle >
     INT REL_ALT             < AC altitude relative to obstacle >

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------------

                     <*** PWILST ENTRY AIRSPACE TYPE ***>

STRUCTURE AIRSPACE

  GROUP maintenance_info

     PTR NXTPWI               < next entry >

     PTR PRVPWI               < previous entry >

     BIT SENT                 < entry has been sent in message >

  GROUP status

     INT AIRSPACE_NO          < identity on stored list of airspace zones >

     BIT END                  < entry not updated this scan >

     BIT FTAT                 < first time advisory transmitted >

  GROUP adv_data

     INT AIRSPACE_TYPE        < restricted airspace or TCA or other
                                 prohibited area >

     CHR IDENTIFIER           < restricted airspace identifier >

ENDSTRUCTURE;
```

```
------------------------------------------------------------------------------------

                      <*** PWILST ENTRY ALEC TYPE ***>
STRUCTURE ALEC


  GROUP maintenance_info

     PTR NXTPWI               < next entry >

     PTR PRVPWI               < previous entry >

     BIT SENT                 < entry has been sent in message >


  GROUP adv_data

     INT ALTITUDE             < altitude echo data >

     BIT CONFIDENCE           < altitude confidence >

     BIT CORRECTED            < pressure corrected altitude >


ENDSTRUCTURE;
```

```
------------------------------------------------------------------------------------

               <*** RESOLUTION PAIR ACKNOWLEDGEMENT (RPA) LIST ***>

STRUCTURE RPALST

  GROUP ac1

     INT DEL                  < delivery status of maneuvers >

     INT ID                   < ID of AC1 >

     INT RES                  < resolution advisory for AC1 >

  GROUP ac2

     LIKE RPALST.ac1

  GROUP ovrhd

     BIT CACRD                < CONFLICT RESOLUTION DATA TASK has run >

     BIT CARN                 < RESOLUTION NOTIFICATION TASK has run >

     BIT SOURCE               < ATARS originated resolutions, else BCAS >

     INT TIME                 < time entry placed on RPA list >

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

                         <*** STATE VECTOR ***>

STRUCTURE SVECT


  GROUP pointers

     PTR ATCREF             < entry in CREFX for ATCRBS AC >

     PTR CTE                < conflict table entry for this AC >

     PTR CTPTR              < conflict table containing this AC's entry >

     PTR NEXTA              < next AC in ATARS sector list thread >

     PTR NEXTS              < next AC in antenna sector list thread >

     PTR NEXTX              < next AC in X_list or EX_list thread >

     PTR PREVX              < previous AC in X_list or EX_list thread >

     PTR PWPTR              < PWILST for AC >

     PTR STKPTR             < stack of AC information (position, velocity,

                               time) for turn rate computation >

     PTR UPMES              < list of last uplinked ATARS messages >


  GROUP horz_tracker_data

     FLT AZP                < predicted next-correlation azimuth >

     INT FIRME              < external firmness level >

     INT FIRMI              < internal firmness level >

     FLT RHOP               < predicted next-correlation range >

     FLT VSQ                < square of horizontal velocity estimate >

     FLT X                  < X position of AC >

     FLT XD                 < X velocity of AC >

     FLT XDE                < external X velocity estimate >

     FLT XDI                < internal X velocity estimate >

     FLT XP                 < external one-scan predicted X position >

     FLT XPI                < internal one-scan predicted X position >

     FLT Y                  < Y position of AC >

     FLT YD                 < Y velocity of AC >

     FLT YDE                < external Y velocity estimate >

     FLT YDI                < internal Y velocity estimate >

     FLT YP                 < external one-scan predicted Y position >

     FLT YPI                < internal one-scan predicted Y position >
```

```
GROUP vert_tracker_data
   INT FIRMZ              < mode C firmness level >
   FLT FIRMZP             < firmness of level occupancy time >
   FLT LOT                < level occupancy time estimate >
   INT SUCNT              < start-up counter to establish track >
   FLT SUMRES             < summed residual used to detect trend in vertical
                             acceleration >
   FLT Z                  < Z position of AC >
   FLT ZD                 < Z velocity of AC >
   FLT ZDP                < external Z velocity estimate >
   FLT ZP                 < external one-scan predicted Z position >
   FLT ZPREV              < previously reported Z position >
   FLT ZS                 < smoothed Z position estimate >

GROUP times
   FLT ALECT              < time last ALEC entry generated >
   FLT OWNT               < time last OWN message generated >
   FLT TD                 < time ATARS message was received by AC >
   FLT TLUPD              < time of last vertical track update >
   FLT TM                 < time of last reported range/azimuth data >
   FLT TMP                < expected time of next local data >
   FLT TMR                < time of current remote data >
   FLT TMZ                < time last mode C report was received >
   FLT TTPRAL             < time of transition to previously reported altitude >
```

```
--------------------------------------------------------------------------------

GROUP flags
   BIT ATIFLG              < track was initialized as ATCRBS >
   BIT ATSS                < AC ready for ATARS service (on XINIT list) >
   BIT CAREQ               < controller alert required in this area >
   BIT CENTR               < AC in center area of ATARS service area >
   BIT CUNC                < AC under ATC control >
   BIT DELFG               < state vector is to be deleted >
   BIT DLOUT               < data link out on most recent scan >
   BIT DRATS               < AC ATARS service dropped but track maintained
                              for domino resolution >
   BIT DRSUR               < AC track dropped by ATARS >
   BIT EXFLG               < X/EX_list indicator >
   BIT HUBFLG              < AC in antenna hub zone >
   BIT INXFL               < new AC on XINIT list >
   BIT LOFL                < this AC has local data >
   BIT MCFLG               < altitude data provided through mode C report
                              and reasonable >
   BIT NULLFG              < a null DABS report is received >
   BIT OSCFL               < BETRAR will be reset after one scan >
   BIT PSTAT               < local site's primary status >
   BIT RMFL                < AC has remote data >
   BIT SMPR                < AC data smoothed, predicted this scan >
   BIT SPIDFG              < signpost flag for coarse screen identification >
   BIT SPRO                < antenna sector processing flag >
   BIT SOLO                < squitter lockout set
   BIT SRVMSK              < report in ATARS/[illegible] surveillance area mask >
   BIT XUPFL               < prevents multiple X/EX_list update >
```

```
--------------------------------------------------------------------------------

GROUP general_values

   INT ACAB              < AC abbreviated field, 9-bit field >
   INT ACAT              < AC area type >
   INT ACLASS            < ATARS class of service >
   INT ACLP              < AC climb performance used in modeling resolution
                            advisory >
   CHR ASSOC             < ID of closest airport >
   INT ATSEQ             < CAS equipage type >
   INT BCASSL            < BCAS sensitivity level >
   INT CODE              < DABS or ATCRBS mode C code >
   INT FAZ               < final approach zone indicator >
   INT FILE              < ATCRBS surveillance file number >
   INT GEOG              < geographical zone 4-bit field >
   INT HMS               < horizontal maneuver status for level tracker >
   INT IND               < unique index of area type 2 >
   FLT OWNHDG            < last heading sent in OWN message >
   FLT OWNTRN            < last turn rate sent in OWN message>
   INT REMRAR            < ID of site whose RAR requested >
   INT RETRAR            < ID of remote site that receives RAR data >
   INT SLREPS            < most recently reported slant range >
   INT SVSID             < AC's ATARS sector ID >
   FLT TERALT            < altitude from terrain map >
   INT TURN              < AC turn status >
   INT TYPE              < AC type status (DABS or ATCRBS) >
   CHR ZPRT              < ID of airport associated with FAZ >
```

```
--------------------------------------------------------------------------------

  GROUP domino_obj_proj
     FLT XDPRJ(4)             < one, two, three and four-scan XD projection after DELAY
                                 time of modeling >
     FLT XPRJ(4)              < one, two, three and four-scan X projection after DELAY
                                 time of modeling >
     FLT YDPRJ(4)             < one, two, three and four-scan YD projection after DELAY
                                 time of modeling >
     FLT YPRJ(4)              < one, two, three and four-scan Y projection after DELAY
                                 time of modeling >
     FLT ZDPRJ                < Z direction velocity for domino object processing >
     FLT ZPRJ(4)              < one, two, three and four-scan Z projection after DELAY
                                 time of modeling >

  GROUP data_block
     BITS REPORT              < DABS or ATCRBS or RADAR report(Tables 3-3, 3-4, 3-5) >

ENDSTRUCTURE;
```

```
------------------------------------------------------------------------------

                         <*** SYSTEM VARIABLES ***>

STRUCTURE SYSVAR


  GROUP time

     FLT CTIME                 < current time >


  GROUP flags

     BIT ATCNMC                < ATC requests non_mode C ATARS target tracking desired >

     BIT ATCROR                < ATC requests radar only target tracking desired >

     BITS STATMSG              < ATC facilities requesting

                                  full ATARS status control messages >


  GROUP failure_info

     BIT BACKUP                < operating in backup mode >

     PTR CTRPTR                < center zone map in use >

     INT FAILED                < identity of failed site >

     PTR MAPPTR                < ATARS service map in use >

     BIT MASTER                < operating in backup-master mode >


  GROUP antenna

     FLT APOS                  < DABS antenna position >

     FLT ARATE                 < DABS antenna rate >


  GROUP general

     INT LOCAL_ID              < local sensor site identification (4 bits) >

ENDSTRUCTURE;
```

4. SURVEILLANCE REPORT AND TRACK PROCESSING

4.1 General Requirements

ATARS surveillance processing may be divided into three main sub-functions: input processing, track processing, and smoothing/prediction. The general division of responsibility is that input processing accepts target reports from the DABS sensor and screens and prepares them for tracking. Track processing maintains a file of system tracks and selects particular target reports for track update. Smoothing and prediction utilizes the selected reports for production of fresh position and velocity updates.

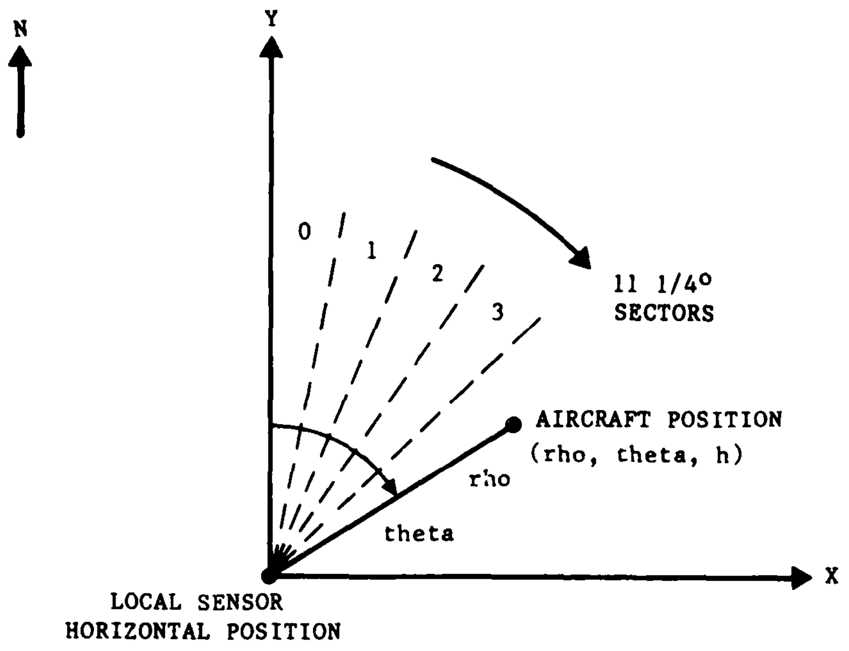
ATARS surveillance processing is required to track those DABS, ATCRBS (mode C and non-mode C equipped) and radar-only aircraft which are being tracked by the local sensor in the ATARS service area and the domino surveillance area. Aircraft outside these areas are not tracked.

4.2 Coordinate Systems

Target reports are received in the rho, theta, h (slant range, azimuth, altitude) system of the DABS sensor. ATARS maintains and uses track estimates in a modified Cartesian x, y, z system. The local sensor lies at the center of the x, y grid with x east and y north (see Figure 4-1).

Mapping between these two systems is based on a flat earth assumption. Under this assumption x, y are considered as the ground plan projection coordinates of an aircraft, while z is identical to the altitude. However, because of the actual curvature of the earth, the x, y, z which are so computed do not exactly correspond to the aircraft position in a physical Cartesian space. Nevertheless, the x, y, z descriptions produced by this formal mapping will be utilized throughout the ATARS processing. The equations of the mapping are shown in the figure.

Geographical corrections take place in rho, theta. Reports selected for track update are coordinate converted to the x, y, z system before using them to smooth the track estimate. The inverse mapping is used to determine a predicted rho, theta for the next correction and for antenna sector update.



TARGET MEASUREMENTS: RANGE rho
AZIMUTH theta
ALTITUDE h

SENSOR ALTITUDE: h_s

SLANT RANGE CORRECTION: $\text{rho}' = \sqrt{\text{rho}^2 - (h - h_s)^2}$

TRACK COORDINATES:

$$X = \text{rho}' \sin(\text{theta})$$
$$Y = \text{rho}' \cos(\text{theta})$$
$$Z = h$$

FIGURE 4-1
ATARS TRACK SECTORIZATION AND COORDINATES

In converting remote reports to local x, y, z coordinates, any method of calculation may be used which accurately accounts for the real spatial geometry. In order to be useful in tracking, the truncation errors in conversion must be kept comparable to the random data error (i.e., approximately = 100 feet or less). The method should be reasonably efficient, but since remote report conversion will only be occasionally required, some complexity can be tolerated.

4.3 Major Files

The ATARS track file is a separate creation of the ATARS function and is not identical, either physically or in content, to the DABS sensor track file. This file (the Central Track Store (CTS)) consists of a block of track slots, of sufficient size to accommodate the maximum track load. Each slot may either be empty or it may contain track information (aircraft State Vector) about a particular aircraft. In addition, a number of slots of known fixed x coordinate distance called signposts provide quick points of entry into the appropriate X-list or EX-list as described in Section 6.1.3. Signposts are bookkeeping aids and are not altered by the tracking programs.

A CTS file is used to transmit information for a particular aircraft throughout the various tasks in the ATARS processing cycle. The central track store contains the following types of information: position, velocity, time, pointers, status and ID indicators, and control flags.

The tracks in CTS must be rapidly accessible in two ways for Report Processing and Track Processing Tasks: on a geographical antenna sector basis and by aircraft ID. 32 fixed azimuth sectors are defined with respect to the local sensor, beginning clockwise from north (see Figure 4-1). Each antenna sector is 11 1/4 degrees wide. Tracks are organized (e.g., by threading) so that the ATARS tracker can efficiently index and process tracks lying in a particular antenna sector. Since the aircraft move, their antenna sector assignments will change. These changes are monitored, and an updated antenna sector list is maintained.

Rapid access of individual tracks through their ATCRBS/radar surveillance file number or DABS ID is accomplished by establishing a cross-reference file for each of these aircraft classes. These files are denoted CREFA and CREFD, respectively, and relate the input code (which may be compressed by hashing) to the corresponding track slot number in CTS. When tracks are dropped or new tracks are started, the cross-reference files are correspondingly updated. These file relationships are indicated in Figure 4-2.

Since the ratio of radar, ATCRBS and DABS track loads is an environmental variable, it is desirable that CTS not be partitioned in any fixed way between these track classes. Procedures for organizing and accessing CTS should remain efficient regardless of this load ratio.

4.4 Report Processing

The ATARS surveillance processing initiates and maintains ATARS tracks on all DABS, ATCRBS (mode C and non-mode C equipped), and radar-only aircraft in the ATARS surveillance area which are being tracked by the DABS sensor. This objective is accomplished by two major tasks: the Report Processing Task, discussed in this section, and the Track Processing Task, discussed in Section 4.5.

Report processing operates on interrupt after all target reports for a new antenna sector have arrived in the surveillance input buffer. The surveillance input consist of target reports from the DABS sensor.

Input data are:

1. A sector header antenna azimuth word
2. Target reports from Surveillance Buffer

The services performed are to:

1. Update current antenna position and rate estimates
2. Screen local reports and reject all reports falling outside the ATARS surveillance area
3. Reject reports according to ATC selection not to process non-mode C or radar-only reports
4. Correlate local and remote reports with tracks through DABS ID or ATCRBS/radar surveillance file number cross-references. Store reports with the State Vectors in CTS
5. Start new DABS, ATCRBS or radar-only tracks
6. Flag tracks for drop (if indicated by sensor)

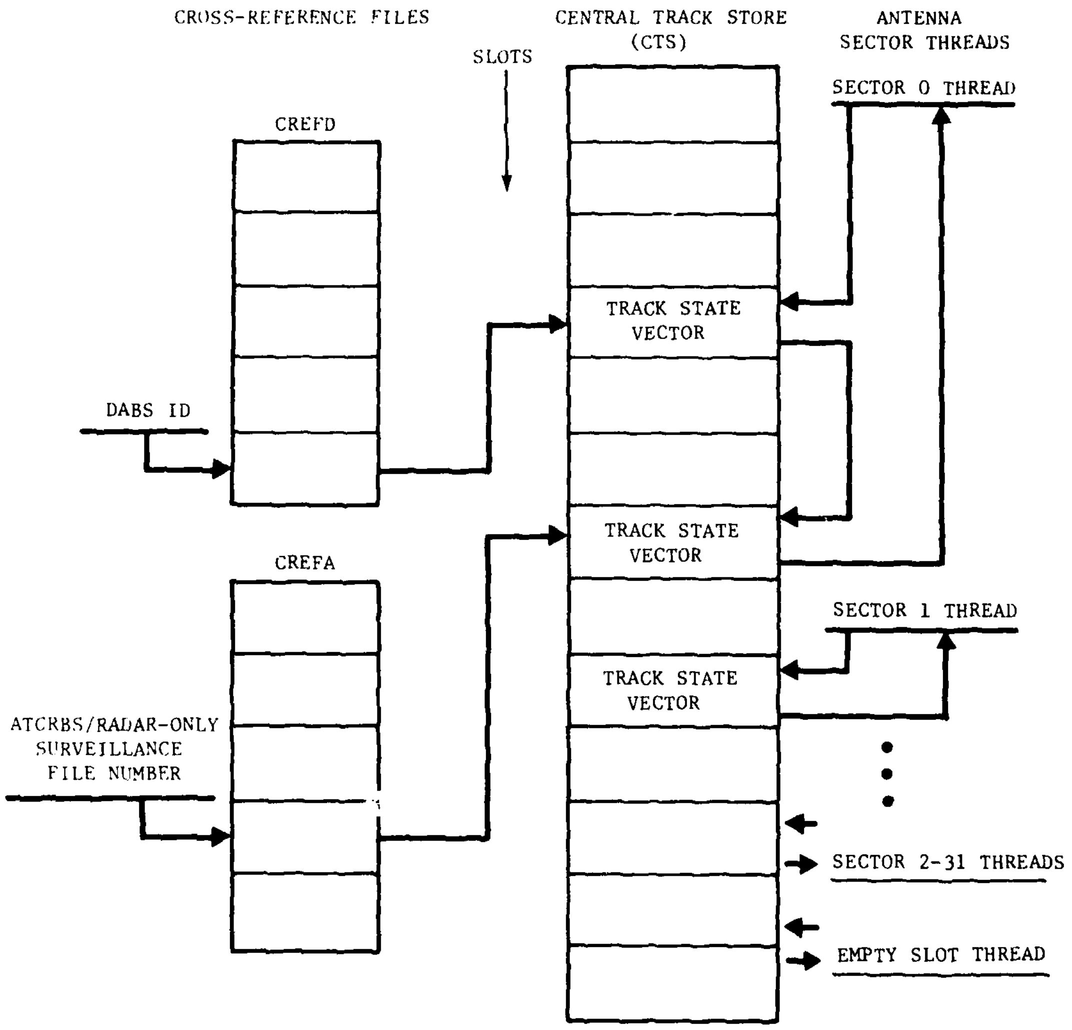


FIGURE 4-2
TRACK FILE STRUCTURE AND ACCESS

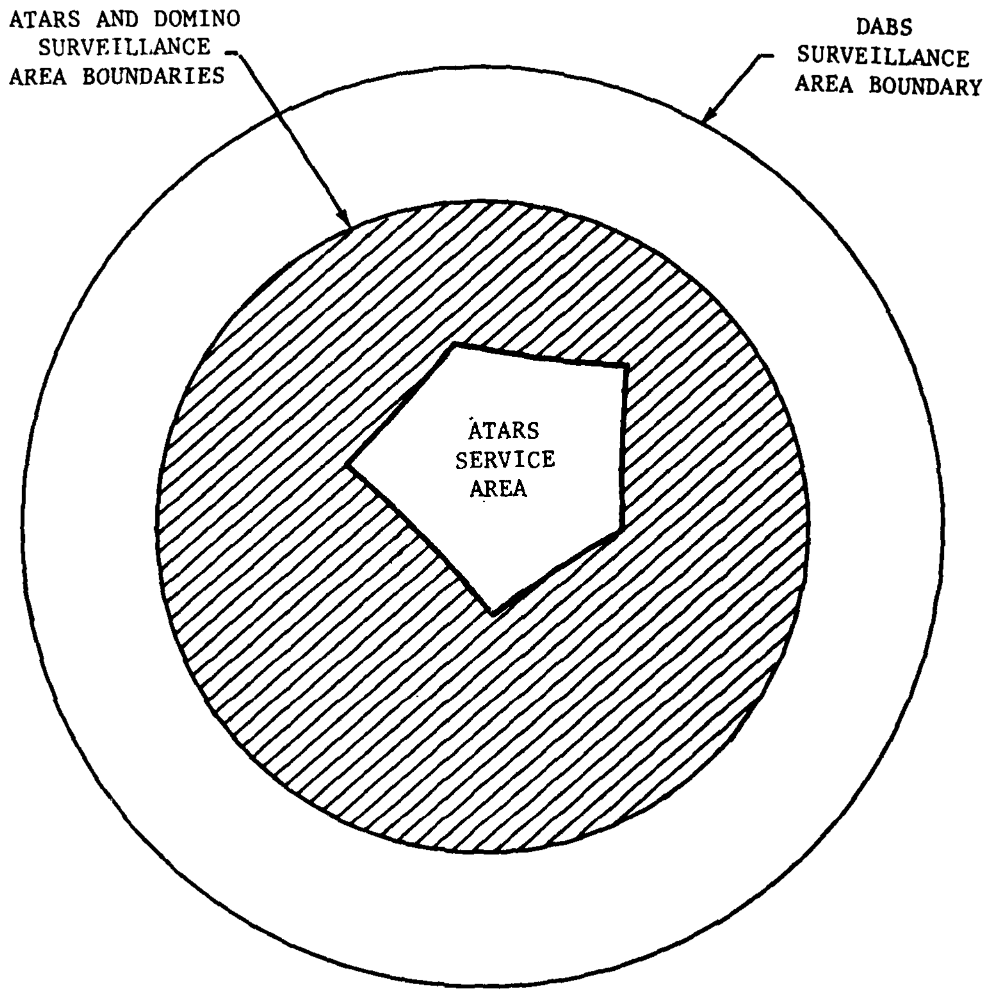
4.4.1 ATARS Surveillance Area

The ATARS service area is subtended by a larger area called the ATARS surveillance area, defined by a rho, theta map. This map is defined as a convex figure encompassing the ATARS/Domino surveillance area as shown in Figure 4-3. Since the ATARS surveillance area is a convex figure, it suffices to define a map with a minimum and maximum rho for each theta (or theta interval). Two maps are required, one for reports above an altitude HZONE and one below this altitude (see Section 6.2.1 and Figure 6-1). In order to pass screening, a local report with altitude data must lie within the area appropriate for that altitude. If no altitude has been measured, then it must lie within at least one of the two map areas. (Remote reports cannot be mapped out efficiently here because their rho, theta are not local coordinates.)

4.4.2 Local Reports

When a local report is within the ATARS surveillance area, the DABS ID or ATCRBS/radar surveillance file number is used to find the associated track in CTS (if any). The cross-reference files CREFD and CREFA provide the required links. If the track's drop flag is already set in CTS, it cannot accept further data and the report is ignored. If the report drop bit is set in the surveillance report, track drop is initiated here. Track drops may also be initiated in track update (Section 4.5.1) when data has been missing too long or the track is out of the ATARS surveillance area. The State Vector Deletion Task performs the final drop action for tracks which have been in ATARS coverage.

When a local report is a null report, the null flag is set to indicate to the Track Processing Task that this particular track is to be treated as a "miss" rather than a "hit." If the diffraction zone bit is set in the report, the track is also processed as a "miss." The local report that is designated as a "hit" must also pass a rho, theta reasonability check of measured vs. predicted coordinates in order to merit further consideration. The report is then stored in the CTS report storage area. A RAR empty report is issued for the aircraft if the proper conditions exist for this message generation. If the site ID has changed for the aircraft, then a check is made to determine if the aircraft is located in the seam area, and so designated. If a pilot request for an altitude echo advisory is received, the proper computations are made and the time of the altitude request is recorded in the track State Vector.


ATARS AND DOMINO SURVEILLANCE AREA BOUNDARIES
DABS SURVEILLANCE AREA BOUNDARY
ATARS SERVICE AREA

DOMINO AREA - PROVIDES COVERAGE AT LEAST AN ADDITIONAL 15 nmi BEYOND THE ATARS SERVICE AREA

FIGURE 4-3
ATARS AND DOMINO SERVICE AREAS

If a new track is to be initialized, this is the sole responsibility of the report processing function. This will occur when a local or late local report is examined which does not have an existing associated track and the report drop bit is not set. Track initialization is started with a single local report.

4.4.3 Remote Reports

If a remote report is a null report, it is rejected at this time. Otherwise, the DABS ID or ATCRBS/radar surveillance file number is used to find the associated track in CTS (if it exists). The cross reference files CREFD and CREFA provide the required links. The remote surveillance report data is then stored in the CTS storage area provided a local report hasn't been received for this aircraft in the last 1/2 scan. The remote flag is set and report time is stored. A RAR empty report is issued for this aircraft if the proper conditions exist for this message generation. If the site ID has changed for the aircraft, then a check is made to dertemine if the aircraft is located in the seam area, and so designated. If a pilot request for an altitude echo advisory is received, the proper computations are made and the time of the altitude request is recorded in the track State Vector.

4.4.4 Track Initialization

An empty track slot, which has been cleared of all previous data, is found in the CTS. Empty tracks are threaded together into their own list using the sector thread mechanism. This slot will hold the new track State Vector. This report's measurement time is determined and then stored. The report coordinates are converted to x, y, z. The initial horizontal prediction estimates for external and internal positions, and velocities are determined and stored. The initial predicted rho, theta search position is computed. The initial internal and external position predictions are set identical to the reported positions. The level velocities are set to a small non-zero value. The turn rate stack is initialized for the track to be used in the x, y smoothing process. The antenna sector in which this track lies is determined and the track is added to the proper antenna sector list (with forward threading only). Various pointers and flags contained in the State Vector are initialized.

With mode C altitude data present, initialization must be performed for the vertical tracker and other ATARS tasks must be notified that valid altitude data is available for this aircraft. Aircraft track type and code information that is unique for either DABS or ATCRBS reports must be stored in the State Vector during track initialization.

4.5 Track Processing

The Track Processing Task performs (1) final elimination of tracks which are not to be serviced by ATARS and (2) Track Update Process. All surveillance reports have been associated with tracks or used to start new tracks in the Report Processing Task. Track processing accepts each report and calls the Track Update Process (Section 4.5.1). All report correlation is accomplished in the DABS sensor and is accepted as being completed by ATARS.

Input data for track processing consists of local or remote reports with remote times of measurement which have been stored with the associated track in CTS by report processing (one report, or none, per track).

Track processing operates once each time the local sensor antenna enters a new antenna sector and processes tracks in particular sectors (relative to the antenna). The sectorization of CTS is accomplished by threading, which is updated as aircraft positions are re-predicted.

The program is organized so that when the antenna enters antenna sector n, all tracks associated with antenna sector n-4 are processed. All other tracks which were input to the Surveillance Buffer during sector n are then processed, (see timing diagram, Figure 4-4). The antenna sector gap from n-4 to n allows time for DABS sensor report processing and sensor to ATARS transmission delays. The maximum delay for surveillance reports is expected to be 3 antenna sectors.

The primary function of track processing is to perform track updating in antenna sector n-4 normally with local data, but if this is missing, to attempt remote data or late local data updates each following antenna sector. This allows timely use of remote reports. Various processing flags and time checks are utilized to prevent too frequent updates or confusion because of ATARS sector changes.

All tracks in antenna sector n-4 whose smooth/predict and antenna sector process flags are not set are processed as a

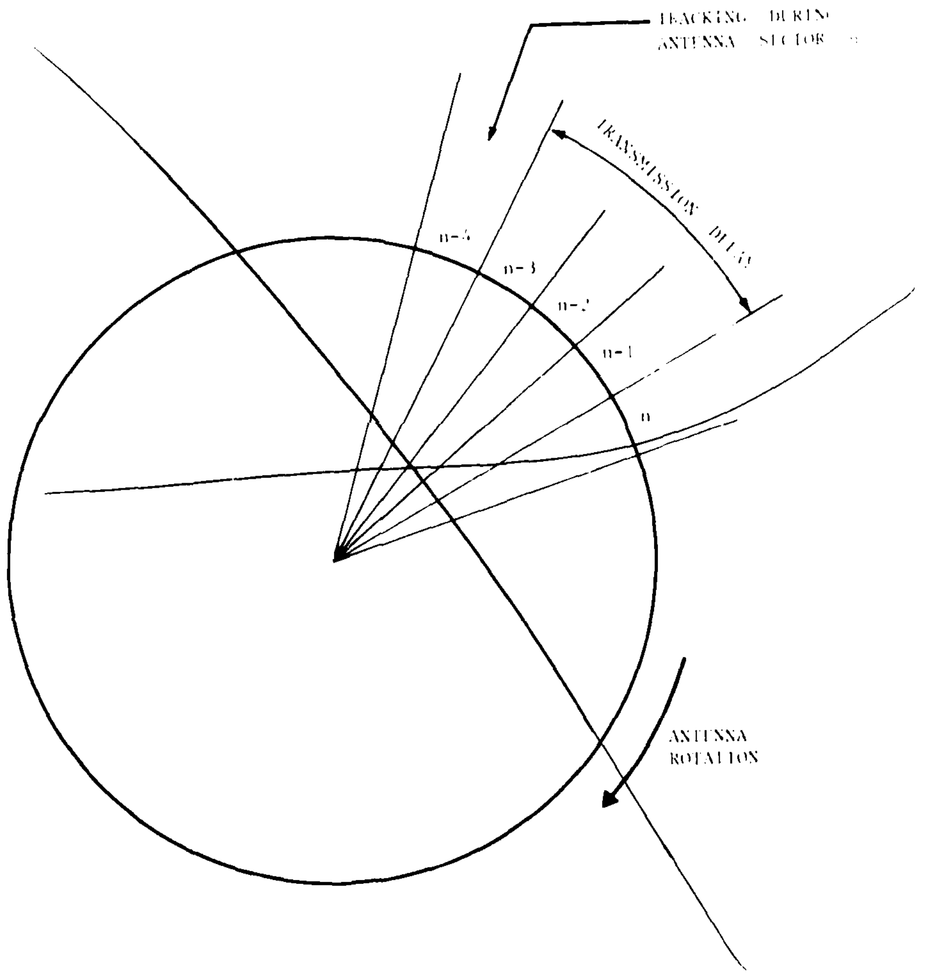


FIGURE 4-4
TRACKER TIMING AND SEQUENCE

group. If a report is not a null report, a track update (hit) is performed, which includes smoothing and prediction. If a null report has been stored, a track update (miss) is performed, which provides prediction only. Since processing for this aircraft has been accomplished, the antenna sector process flag is set in the State Vector.

All the remaining tracks in the buffer, both late local and remote reports, are processed as a second group. If the smooth/predict and the antenna sector process flags are clear, then the remote data is accepted and a track update (hit) is performed which includes smoothing and prediction.

4.5.1 Track Update

The Track Update Process has two entry points. One is used to process a track after a report has been selected (a hit); the other entry is used to process the track in final prediction only if the track report is a null (a miss). The update process uses smoothing and prediction and performs other State Vector update functions as well. Accessory bookkeeping operations affect the CREFA and CREFD files.

4.5.1.1 Selected Report (Hit) Updating

The measurement time is computed for the current local and remote reports. This time is computed from the report azimuth and the antenna position/rate estimates. The report time is set into the current report time in the CTS.

The previous predictions for this track should be corrected to the current measurement time, if necessary. The predictions are for an anticipated time of local data measurement. Thus, if this report is local, no correction will ordinarily be required. However, remote data will be measured at quite different times, and the previous predictions must be corrected. Correction is accomplished by shifting the x, y and z predicted coordinates by the time difference (TMP - TMR) times the internal x, y and z velocity estimates. Both internal and external predictions will be affected (see Section 4.5.3).

Rho, theta data is coordinate converted to the local x, y system. Conversion of local data involves slant range correction as indicated in Figure 4-1. For remote reports a full remote to local system conversion must be used. These conversions utilize the altitude measurement if present. Otherwise, the estimated altitude is used. In cases where neither is available, a nominal value of ZNOM will be assumed. The slant range for both local and remote surveillance reports must be transferred into SLREPS before any coordinate conversions are made to the data.

For local reports the fact that smoothing/prediction is accomplished is noted in the State Vector. If the report is a DABS report, the identity of the remote site is checked in the RAR. If it is determined that the local site has coverage, a message is sent to this ATARS site to stop sending RAR data to this aircraft and inform the local DABS sensor to start ATARS service for this aircraft. If it is determined that the remote site identified in the RAR has control, then that remote site is where data may be requested.

For a remote report a check is made of the measured x, y position against the external prediction (corrected) for reasonableness. If the report is not reasonable, no update is made to position. If the aircraft is in the cone of silence, it is being afforded ATARS service and is ATARS equipped and the remote site is identified from which data will be requested, a message is sent to the site requesting RAR data for this aircraft. Sent along with this message is own-site ATARS identification and the directive for the remote site to set its one scan flag (OSCFL). The local DABS sensor is informed to stop ATARS service for this aircraft.

After x, y smoothing has been performed for the aircraft, z smoothing and prediction with the ATARS vertical tracker (Section 4.5.3.3) is performed. The controlled/uncontrolled status of the aircraft is set according to the new DABS report. The local and remote flag indicators in CTS are also cleared.

4.5.1.2 Null Report (Miss) Updating

The flag which indicates the local sensor has lost data link contact with this aircraft is set in the State Vector. If present time minus TM is greater than TDROP, then ATARS surveillance is dropped for this aircraft. The aircraft is placed on the deletion list and unlinked from the X/EX-list.

4.5.1.3 Updating for Selected and Null Reports

If ATARS surveillance drop is indicated and the aircraft is not on the X/EX-list, the track is dropped and its cross-reference links (CREFA or CREFD) are erased. For those aircraft designated as ATCRBS or radar, the State Vector Deletion Task has responsibility for the drop but the track cross-reference (CREFA) is deleted. Dropping a track here consists of simply re-threading it onto the empty list.

For tracks whose ATARS surveillance drop is not indicated, horizontal track positions are predicted to the estimated time of next local data using the ATARS prediction algorithm. Azimuth and range are computed from the predicted position values and stored with the prediction time.

The track is tested for ATARS qualification with the use of a firmness control. (NOTE: Firmness control is discussed in Section 4.5.2.)

If the track is qualified for ATARS service, see if it is on the X-list or the EX-list. If the track is not in either list, place it in the XINIT List for the Aircraft Update Processing Task by use of NEXTX. If the track is on either list and ATARS service is not being afforded this aircraft, test the track position estimates (external prediction; XP, YP, ZP) to determine whether the track lies within the ATARS service area. This is accomplished by a geographical area determination which utilizes an x, y masking procedure. This x, y mask is defined as the ATARS mask service area as shown in Figure 4-3.

If the track is within the area, ATARS service is given to the aircraft. The tracks are placed on the appropriate X-list or EX-list in the aircraft update task. If the aircraft is ATARS equipped, a message is sent to the DABS sensor to start ATARS service for this track. For all DABS tracks, the primary/secondary status of the track is recorded in the State Vector.

For the track not qualified for ATARS service, see if the track is on the X-list or the EX-list. If this is true, drop ATARS service is initiated, and the aircraft is placed on the deletion list and unlinked from the X-list or the EX-list.

4.5.2 Firmness Control

The ATARS tracking system uses the method of firmness controlled smoothing parameters (see Table 4-1). Separate values are given for beacon and radar-only reports. Firmness values for each track are adjusted in accordance with its record of correlation success (see Table 4-2). The firmness table construction and

TABLE 4-1

HORIZONTAL SMOOTHING PARAMETERS VS. FIRMNESS

FIRM	ALFA		BETA		THK
	Beacon	Radar	Beacon	Radar	
0	1.000	1.000	0.000	0.000	-
1	1.000	1.000	1.000	1.000	LPV[1]
2	1.000	1.000	1.000	1.000	LPV
3	0.833	0.780	0.700	0.650	3.60
4	0.700	0.595	0.409	0.360	2.00
5	0.600	0.475	0.270	0.220	1.50
6	0.524	0.395	0.192	0.145	1.26
7	0.464	0.345	0.144	0.093	1.12
8	0.417	0.310	0.112	0.058	1.03
9	0.400	0.300	0.100	0.050	1.00

[1]LPV indicates a very large positive value which disables turn detection.

TABLE 4-2

FIRMNESS TABLE STEP CONTROL

FIRM (Current Value)	HORIZONTAL AND VERTICAL CORRELATION SUCCESS FIRM (Next Value)	HORIZONTAL CORRELATION FAILURE FIRM (Next Value)	VERTICAL CORRELATION FAILURE FIRM (Next Value)
0	2	0	0
1	3	1	0
2	3	2	1
3	4	2	2
4	5	2	3
5	6	3	4
6	7	4	5
7	8	5	6
8	9	6	7
9	9	7	8

Note: Additional maximum limits may be applied to firmness levels.

AD-A104 147 MITRE CORP MCLEAN VA METREK DIV F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-1 FAA-RD-81-45-1 NL

2 OF 5
AD
A104147

use for the ATARS tracking system differs from its Augmented ARTS III counterpart in several ways. The following features are present.

1. Fewer levels have been assigned and the alpha, beta smoothing parameter values of these levels have been altered.

2. Together with alpha and beta an additional threshold parameter, THK, has been added to adjust ATARS cross-track smoothing thresholds. (The notation (THK = LPV) means that the ATARS turn detection should be disabled.)

3. Three firmness values, FIRMI, FIRME and FIRMZ, are maintained on each track. Each uses a lookup table to select parameters for internal x, y, for external X, Y, or for altitude smoothing. FIRMI is used for THK lookup.

4. These firmness values step up or down depending on the success or failure of attempts to correlate on a particular scan. The adjustments are made after the previous firmness levels have been used for lookup. Nominal adjustments, which are common to all three firmness values, are further subject to absolute assigned maximum firmness limits. Nominally,

   $FIRMI_{max} = 9$
   $FIRME_{max} = 4$
   $FIRMZ_{max} = 9$

   These values cannot be exceeded during stepping. Note that a successful rho, theta correlation does not necessarily result in a successful altitude correlation so that FIRMZ does not necessarily increase with FIRMI and FIRME. Altitude reports may be missing.

5. When a track is initiated, FIRMI and FIRME are inserted at level 1. Similarly, if altitude data on this new track has been received, FIRMZ is also initialized at 1. If no altitude data has been received, set FIRMZ = 0.

6. A track is terminated when the time interval between the current time and the time of the last successful horizontal reply (represented by TM) exceeds the threshold value TDROP.

7. All turning track firmness levels used in ARTS III have been deleted.

4.5.3 Track Estimation

Track estimation consists of two processes: smoothing, in which a track's current data is combined with a previously made prediction to achieve an improved estimate of the current state, and prediction, in which the current smoothed state is extrapolated one scan ahead to assist correlation and prepare for the next smoothing step. Figure 4-5 illustrates turn sensing, smoothing and prediction based on the current positions and velocities.

Smoothing and prediction processes are called from the Track Update Process. The essential data communicated to the smoothing algorithm is:

a. The track to be processed
b. The report to be used for smoothing
c. Estimates of antenna position and rate

The prediction algorithm requires only the first and third items. Both algorithms modify the contents of the track State Vector. In both smoothing and prediction, reference is made to internal and external position and velocity coordinates. Internal refers to those positions and velocities used internal to the Track Processing Task. External includes all positions and velocities used elsewhere in the ATARS tasks but generated in the Track Processing Task.

4.5.3.1 Horizontal Smoothing

The ATARS horizontal smoothing process is designed to provide improved knowledge of aircraft heading during turns and vertical velocity information for the level-occupancy tracker (Section 4.5.3.3). Thus, ATARS conflict detection and conflict resolution may be performed more effectively for maneuvering aircraft than has been possible with previous algorithms.

Horizontal (x, y) positions and velocities are smoothed by the well-known alpha, beta technique until a maneuver is sensed.

The cross-track deviation (data report distance from the line of the predicted track vector) is compared with an assigned threshold to detect a turn. If a Horizontal Maneuver Status (HMS) indicates a turn, the turn rate (W) is computed from a modified cross-track deviation computed from the oldest of three previous smoothed positions and velocities. Figure 4-6 illustrates turn

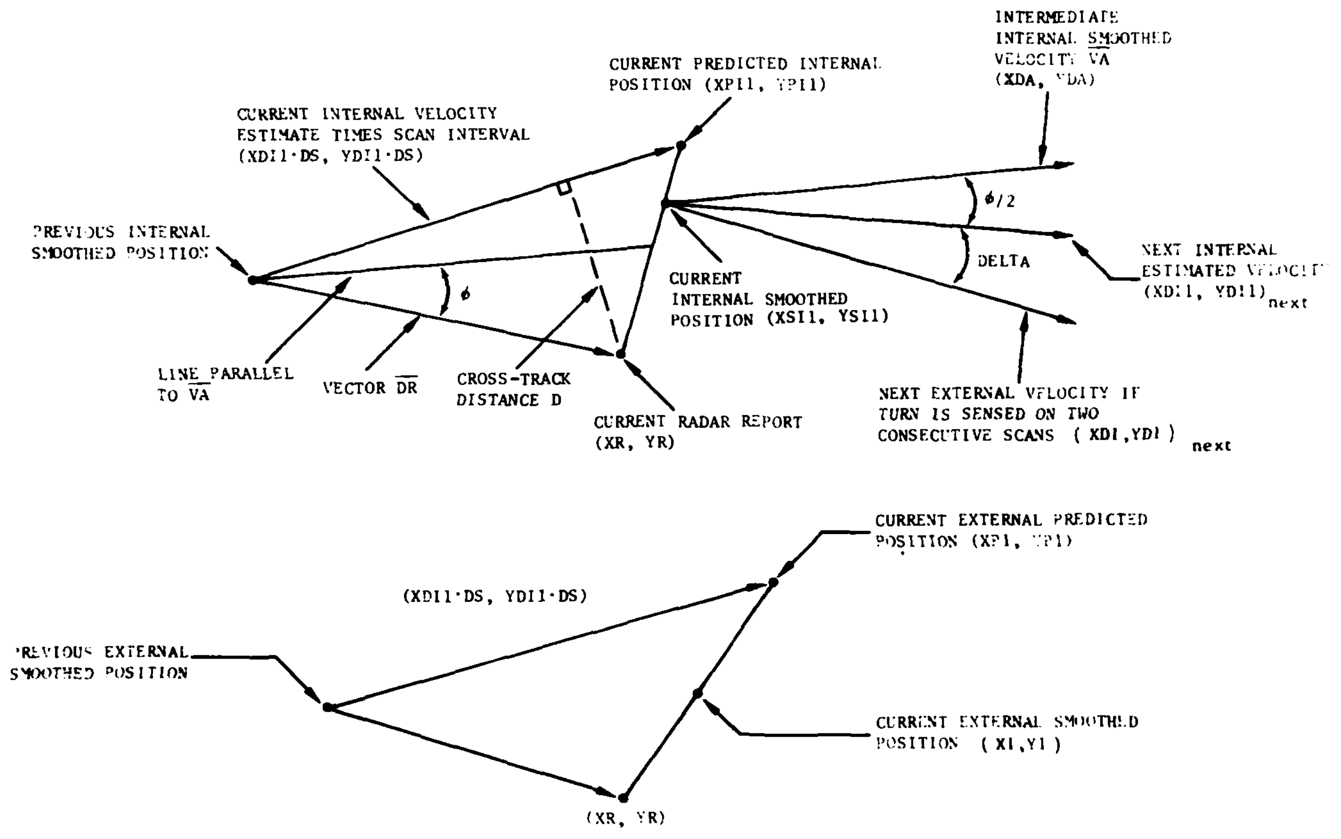

CURRENT PREDICTED INTERNAL POSITION (XPI1, YPI1)
INTERMEDIATE INTERNAL SMOOTHED VELOCITY VA (XDA, YDA)
CURRENT INTERNAL VELOCITY ESTIMATE TIMES SCAN INTERVAL (XDI1·DS, YDI1·DS)
ϕ/2
PREVIOUS INTERNAL SMOOTHED POSITION
NEXT INTERNAL ESTIMATED VELOCITY (XDI1, YDI1)next
DELTA
CURRENT INTERNAL SMOOTHED POSITION (XSI1, YSI1)
ϕ
LINE PARALLEL TO VA
VECTOR DR
CROSS-TRACK DISTANCE D
CURRENT RADAR REPORT (XR, YR)
NEXT EXTERNAL VELOCITY IF TURN IS SENSED ON TWO CONSECUTIVE SCANS (XDI, YDI) next
CURRENT EXTERNAL PREDICTED POSITION (XPI, YPI)
(XDI1·DS, YDI1·DS)
PREVIOUS EXTERNAL SMOOTHED POSITION
CURRENT EXTERNAL SMOOTHED POSITION (XI, YI)
(XR, YR)

FIGURE 4-5
VARIABLES AND CONSTRUCTIONS FOR ATARS X,Y TRACK SMOOTHING

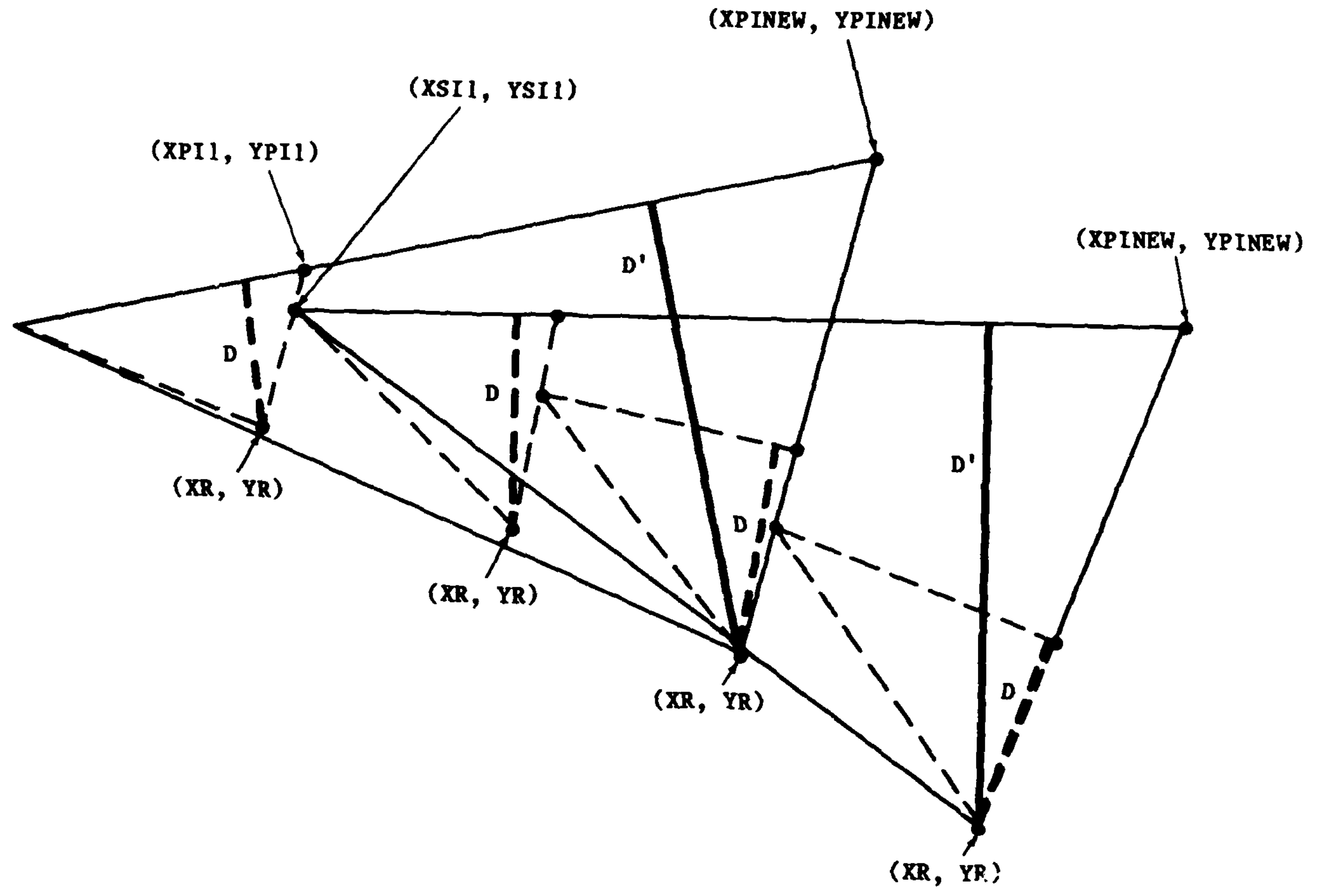


FIGURE 4-6
PREDICTION AND CROSS-TRACK DEVIATION
USING HISTORICAL POSITIONS AND VELOCITIES

rate computation (in relation to Figure 4-5) using historical positions and velocities. Figure 4-7 illustrates the turn rate computations used for the advisory service in ATARS (see Own Message in Section 16.1.3.2). Maneuvering tracks are smoothed by a special method which includes track-oriented geometric calculations. The following implementation avoids use of trigonometric functions wherever possible.

The horizontal smoothing operations are controlled by two important CTS track firmness parameters FIRMI and FIRME which are maintained by the correlation program in accord with the record of correlation successes and failures. Table 4-1 shows how the various smoothing process parameters vary with these firmness values (denoted generically by FIRM). (The level zero is restricted to FIRMZ).

Before the X, Y Smoothing Process is started, the measured range and azimuth are converted to Cartesian XR, YR components. CTS contains the predicted internal position XPI1, YPI1, and velocity, XDI1, YDI1, estimates for the current predicted data time, TMP1. CTS also contains up to three previous smoothed internal positions, velocities, old data times, and the special predicted positions used for the turn rate. For an update with local data, TMP1 is sufficiently close to the true measurement time that these predictions may be directly compared with the measurements. For an update with remote data, the predictions XPI1, YPI1 are first corrected to the remote measurement time by using the velocities XDI1, YDI1 and applying the time difference TMR1-TMP1.

Several quantities to be used in the computation of the turn rate are initialized in the Track Initialization Process, and are available in the State Vector. These variables are updated in the X, Y Smoothing and Prediction Processes. These parameters are outlined below.

Last predicted internal velocity:

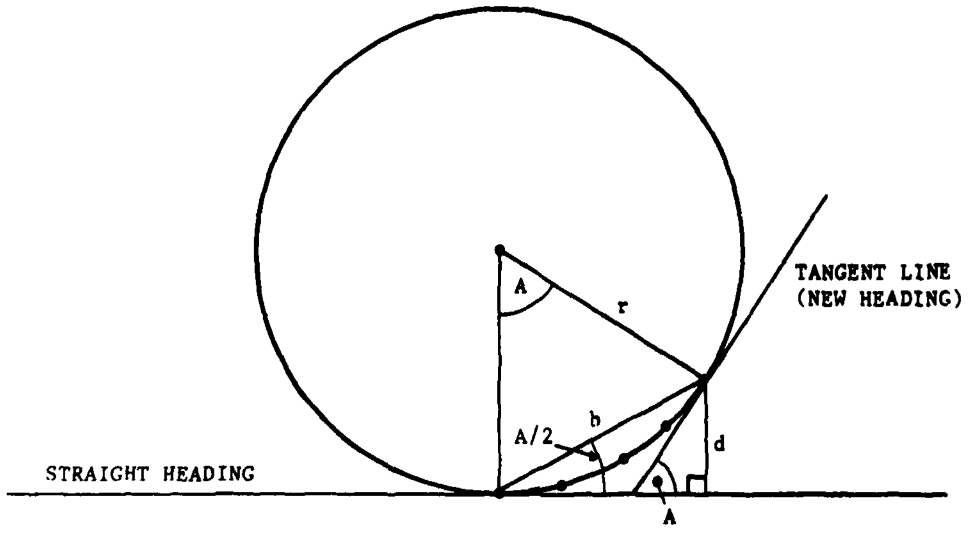
XDIOLD = XDI1
YDIOLD = YDI1

Last predicted internal position:

XPINEW = XPI1
YPINEW = YPI1

Time on the stack:

ST = 0



$\text{SIN } A = b/r$

$b = 2r \text{ SIN } A/2$

$d = b \text{ SIN } A/2$

$= 2r \text{ SIN}^2 A/2$

approx. $= 2r\, A^2/4$ FOR SMALL A

$A = wt$ t = TIME FOR n SCANS

$rA = vt$ v = VELOCITY

$d = 2(vt)wt/4$

$= 1/2\ vwt^2$

$w = 2d/vt^2$ TURN RATE

SUBSTITUTING FOR d AND v, THE TURN RATE EQUATION IS:

$W = 2CTDA/(XDIOLD^2+YDIOLD^2) \quad * \ (DT+ST)^2$

FIGURE 4-7
TURN RATE COMPUTATION FOR ADVISORY SERVICE

The stack time is initialized in the Track Initialization Process and stored in CTS. The stack time is computed each scan in the Track Processing Task.

Then define the internal deviation vector DI as:

$$\overline{DI} = \begin{bmatrix} DIX \\ DIY \end{bmatrix} = \begin{bmatrix} XR - XPI1 \\ YR - YPI1 \end{bmatrix} \quad \text{and}$$

$$\overline{CTDI} = \begin{bmatrix} CTDIX \\ CTDIY \end{bmatrix} = \begin{bmatrix} XR - XPINEW \\ YR - YPINEW \end{bmatrix}$$

Let the elapsed time since the last X, Y data was received on this track be DT. This time difference is calculated from the last data time, TM1, stored in CTS and the new data time. New data time for local and remote reports are determined here from the measured azimuth and the antenna motion estimates.

Then ALFA, BETA are selected through FIRMI, and the smoothing equations produce the intermediate estimated coordinates designated XA, YA, XDA, YDA as follows:

XA = XPI1 + ALFA * DIX

YA = YPI1 + ALFA * DIY

XDA = XDI1 + BETA * DIX/DT

YDA = YDI1 + BETA * DIY/DT

The next step is to sense for turns. Let

A = DIX * YDI1 - DIY * XDI1

CTDA = CTDIX * YDIOLD - CTDIY * XDIOLD

S = Sign (A)

B = $XDI1^2 + YDI1^2$

The cross-track distance, D, used for turn sensing is

D = A/SQRT(B)

The square root can be avoided by dealing with the square of this distance, D2.

$$D2 = A^2/B$$

Let D2TH be a threshold by which D2 is measured to sense a turn. Then, if D2 .GT. D2TH and S is negative, a left turn is sensed. If D2 .GT. D2TH and S is positive, a right turn is sensed.

The threshold is computed as a function of track range, speed, orientation and the data source used for this update. It is further modified by the factor THK which depends on FIRMI. The calculation of D2TH is accomplished through two intermediate quantitites, DTHA and DTHB:

$$D2TH = THK * (DTHA + DTHB * (XA * XDA + YA * YDA)^2/(V2A * R2A))$$

where

$$R2A = XA^2 + YA^2$$

$$V2A = XDA^2 + YDA^2$$

Physically, DTHA is the square of the threshold which is appropriate for testing the radial deviations of a track moving tangentially to the radar. DTHA + DTHB is the square of the threshold appropriate for testing tangential deviations of a track moving radially. The factor multiplying DTHB is the square of the cosine of the angle between the track direction and the radius vector from the radar. Since the predicted range is available from CTS, it may also be used with sufficient accuracy in place of the R2A computation above. The quantities, DTHA and DTHB, are determined from sensor error standard deviations and track speed by the empirical formulas:

$$DTHA = (3.1 * STDA + 1.35 * SQRT\ (V2A))^2$$

$$DTHB = (3.1 * STDB + 1.35 * SQRT\ (V2A))^2 - DTHA$$

The speed estimate, SQRT(V2A), must be expressed in knots for this calculation when the other quantities are in feet.

The sensor error parameters, STDA and STDB, are, respectively, the radial and tangential data error standard deviations as specified in Reference 1. A typical parameter set for each data source is listed in Table 4-3. Tangential parameters generally depend on the track-range, SQRT(R2A). Since remote data is not oriented conveniently in the local sensor system, a pessimistic, isotropic assignment is made.

TABLE 4-3

THRESHOLD PARAMETERS IN FEET

SOURCE	STDA	STDB[1]
Local DABS Beacon	150	.002 * SQRT (R2A)
Local ATCRBS Beacon	180	.002 * SQRT (R2A)
Local Radar	215	.004 * SQRT (R2A)
Remote Beacon	500	500

[1]The range, SQRT(R2A), should be expressed in feet.

When a turn is sensed, a correction in the direction of the sensed turn is made in the heading of the aircraft. Let DR be the vector

$$\overline{DR} = \begin{bmatrix} DRX \\ DRY \end{bmatrix} = \begin{bmatrix} XDI1 * DT + DIX \\ YDI1 * DT + DIY \end{bmatrix}$$

and VA the vector

$$\overline{VA} = \begin{bmatrix} XDA \\ YDA \end{bmatrix}$$

The magnitude of this correction is half of the angle between vectors DR and VA, except when this angle exceeds some threshold, TTH, in which case the correction is limited to TTH/2. (It is assumed that the parameter, TTH will be less than 90^o). Let phi be the angle between vectors DR and VA and let CT2 = $\cos^2$(TTH).

Define

$$C = DRX * XDA + DRY * YDA$$

$$P = Sign(C)$$

Then

$$CP2 = \frac{C^2}{(DRX^2 + DRY^2) * V2A}$$

Define

$$CP = SQRT(CP2)$$

Let sin (abs(phi)/2) = SPD2 and cos (abs(phi)/2) = CPD2. Where abs means the absolute value of the parameter.

$$SPD2 = SQRT\ ((1-CP)/2)$$

$$CPD2 = SQRT\ ((1+CP)/2)$$

Let delta theta be the absolute value of the heading correction and let SDT = sin (delta theta) and CDT = cos (delta theta). Let STD2 = sin(TTH/2) and CTD2 = cos(TTH/2). Then,

$$\left.\begin{array}{l} SDT = STD2 \\ CDT = CTD2 \end{array}\right\} \quad \text{if } P * CP2 \text{ .LE. } CT2$$

$$\left.\begin{array}{l} SDT = SPD2 \\ CDT = CPD2 \end{array}\right\} \quad \text{otherwise}$$

The new estimated internal velocity coordinates are:

$$\left.\begin{array}{l} XDI1_{next} = XDA * CDT + S * YDA * SDT \\ YDI1_{next} = YDA * CDT - S * XDA * SDT \end{array}\right\} \quad \text{if } D2 \text{ .GT. } D2TH$$

$$\left.\begin{array}{l} XDI1_{next} = XDA \\ YDI1_{next} = YDA \end{array}\right\} \quad \text{if } D2 \text{ .LE. } D2TH$$

When a turn has been sensed in the same direction on two consecutive updates, an additional heading correction of magnitude, DELTA, is applied in the direction of turn. Let SDEL = sin(DELTA) and CDEL = cos(DELTA).

If a turn is sensed on two consecutive updates, the final external velocity coordinates used as a source of the coordinates in ATARS detection and resolution are:

$$XDE1_{next} = XDI1_{next} * CDEL + S * YDI1_{next} * SDEL$$

$$YDE1_{next} = YDI1_{next} * CDEL - S * XDI1_{next} * SDEL$$

For all other cases:

$$XDE1_{next} = XDI1_{next}$$

$YDE1_{next} = YDI1_{next}$

Note that the DELTA correction affects the data used for detection and resolution, but does not influence the internal tracker velocities, and, hence is not propagated into the future.

The smoothed internal position estimates are:

$XSI1 = XA$

$YSI1 = YA$

XSI1 and YSI1 are local variables and are processed further by prediction which occurs later.

The horizontal maneuver status indicator, HMS1, stores the turn indication for use on the next update. It is set to zero during track prediction if the predicted time of the miss data is later than THMS beyond the last data time.

Let XP1, YP1 be the external estimates used in ATARS. These are position predictions for the current data time, but distinct and separate from the internal estimates. Define the external deviation vector DE as:

$$\overline{DE} = \begin{bmatrix} DEX \\ DEY \end{bmatrix} = \begin{bmatrix} XR - XP1 \\ YR - YP1 \end{bmatrix}$$

Here, too, for remote data a preliminary correction is applied to XP1, YP1 using XDI1, YDI1 and the time difference TMR1 - TMP1.

After ALFA is selected through FIRME, the smoothed estimates XS1, YS1 are produced by the operation:

$XS1 = XP1 + ALFA * DEX$

$YS1 = YP1 + ALFA * DEY$

XS1 and YS1 are local variables and are processed further by prediction which occurs later.

Both internal and external position estimates are propogated into the future, but positions predicted for turn rate computations (see Figure 4-7) are completely independent and are not propagated into the future. The reason for these two types of estimates is that the internal positions are designed to provide effective turn sensing while the external estimates provide more accurate actual positions for ATARS threat detection and resolution.

The Detect Task and resolution advisories evaluation logic use the turn status as determined by the following logic. The turn sensing algorithm has seven states (noted in Table 4-4) representing different degrees of confidence that the aircraft is actually turning. The cross-track deviation (D2) is checked against two thresholds (TH1, TH2) which are computed from the equations shown on Table 4-5. One threshold is small enough so that the probability of a missed alarm is small and there is little delay in detecting a turn. The other threshold is large enough so that the probability of a false alarm or wrong alarm is minimized (see Reference 6). The transition diagram for the turn sensing states is summarized in Table 4-6.

4.5.3.2 Horizontal Prediction

When a track receives data, prediction is done just after smoothing. The various smoothed estimates are projected ahead to the next local correlation time using the appropriate smoothed velocities. The velocities are not modified. When a track receives no data, the previous predictions are treated as if they were new smoothed estimates and are predicted again. In this case, prediction is accomplished during first pass processing in the tracking module.

Let DS be the estimated time to the next local report. Then,

$$XDI1 = XDI1_{next}$$
$$YDI1 = YDI1_{next}$$

$$XPI1 = XSI1 + DS * XDI1$$
$$YPI1 = YSI1 + DS * YDI1$$

$$XP1 = XS1 + DS * XDI1$$
$$YP1 = YS1 + DS * YDI1$$

To compute XPINEW, YPINEW, use time at stack top minus time at stack bottom to compute time on the stack (ST). Use position and velocity on the bottom of the stack, ST, and DS to predict the next position.

TABLE 4-4

TURN SENSING STATES

VALUE OF (TURN)	DEFINITION
$STRNGLFT	We are very confident that the aircraft is turning left.
$WKLFT	We are slightly confident that the aircraft is turning left.
$STRAIGHT	We are very confident that the aircraft is going straight.
$WKRGT	We are slightly confident that the aircraft is turning right.
$STRNGRGT	We are very confident that the aircraft is turning right.
$HUHMINUS	We are uncertain about the aircraft's turn status.
$HUHPLUS	We are uncertain about the aircraft's turn status.

TABLE 4-5

EQUATIONS FOR TURN SENSING

$$C2T = \frac{(XA * XDA + YA * YDA)^2}{V2A * R2A}$$

$$CRNG1 = (TRKW1 * STDA + TRKW2 * SQRT\ (V2A))^2$$

$$CAZ1 = (TRKW1 * STDB + TRKW2 * SQRT\ (V2A))^2 - CRNG1$$

$$TH1 = THK * (CRNG1 + CAZ1 * C2T)$$

$$CRNG2 = (TRKS1 * STDA + TRKS2 * SQRT\ (V2A))^2$$

$$CAZ2 = (TRKS1 * STDB + TRKS2 * SQRT\ (V2A))^2 - CRNG2$$

$$TH2 = THK * (CRNG2 + CAZ2 * CZT)$$

TABLE 4-6

TRANSITION DIAGRAM FOR TURN SENSING

PREVIOUS VALUE OF (TURN)	NEXT VALUE OF (TURN)				
	STRONG LEFT (IHIT=-2)	WEAK LEFT (IHIT=-1)	STRAIGHT (IHIT=0)	WEAK RIGHT (IHIT=+1)	STRONG RIGHT (IHIT=+2)
$STRNGLFT	$STRNGLFT	$WKLFT	$STRAIGHT	$HUHPLUS	$HUHPLUS
$WKLFT	$STRNGLFT	$WKLFT	$STRAIGHT	$HUHPLUS	$HUHPLUS
$STRAIGHT	$WKLFT	$WKLFT	$STRAIGHT	$WKRGT	$WKRGT
$WKRGT	$HUHMINUS	$HUHMINUS	$STRAIGHT	$WKRGT	$STRNGRGT
$STRNGRGT	$HUHMINUS	$HUHMINUS	$STRAIGHT	$WKRGT	$STRNGRGT
$HUHMINUS	$WKLFT	$WKLFT	$STRAIGHT	$HUHPLUS	$HUHPLUS
$HUHPLUS	$HUHMINUS	$HUHMINUS	$STRAIGHT	$WKRGT	$WKRGT

Note that external X, Y positions are predicted using internal velocities. This is done to reduce the possible perturbations caused by false turn detections.

In order to prepare for the next correlation, the predicted range (RHOP1) and azimuth (AZP1) are calculated from the external predictions and stored in the track file.

$$RHOP1_{next} = SQRT\ (XP1^2 + YP1^2 + ZP1^2)$$

$$AZP1_{next} = \tan^{-1}\ (XP1/YP1)\ \text{(with quadrant determination)}$$

For an update with local data, DS is nominally the estimated scan time of the antenna.

$$DS_{scan} = \frac{360^{o}}{ARATE}$$

The tangential motion of a track near the antenna may require a correction of this value. A method for determining whether correction is needed, is to find ($AZP1_{next}$ - AZP1) and then the increment of extra scan time which this predicted azimuth change requires. The extra time, DDS, is a correction of DS_{scan} and may be positive or negative. If the magnitude of DDS is TDDS or greater, the predictions are recalculated with the exact DS, i.e.,

$$DS = DS_{scan} + DDS$$

For an update with remote data,

$$DS = TMP1 - TM1 + \begin{cases} DS_{scan} + DDS & \text{Loop 1} \\ 0 & \text{Loop 2} \end{cases}$$

The time for which the predictions are made, $TMP1_{next}$, is calculated and stored in the State Vector.

For a hit:

$$TMP1_{next} = TM1 + DS$$

For a miss:

$$TMP1_{next} = TMP1 + DS$$

(TM1 is the time of measurement of the current report, which replaced the old time after the call to the vertical tracker).

4.5.3.3 Vertical Tracker

The level-occupancy vertical tracker, developed by Lincoln Laboratories (see Reference 12), is replacing the older alpha-beta vertical tracker used in the previous ATARS design. The level-occupancy vertical tracker shows a much improved response to swift vertical manuevers, which was a deficiency in the older tracker. The algorithm for the vertical tracker is taken directly from Reference 12 and translated into the pseudocode shown in Section 4.6. The CTS is expanded to include the new quantities necessary to implement this new vertical tracker and the correct initialization is obtained from that provided in Reference 5.

4.5.4 Supporting Routines

This section provides a list of some common routines required by the ATARS tracker. Particular algorithms are outlined in selected cases. Track processing control flags are briefly summarized and their utility noted.

Required routines of particular importance are the following:

1. Antenna Azimuth Position/Rate Estimation

This routine is called by the Report Processing Task. The input consists of an antenna azimuth position from the header word supplied with each antenna sector's reports in the Surveillance Buffer. An azimuth is received each antenna sector, whether or not there are accompanying target reports.

It is also necessary to read the ATARS real-time clock at the time the azimuth is extracted.

The antenna estimate is embodied and stored in three variables APOS, ATIME, ARATE. Let the new input azimuth be ANAZ and the clock time CTIME. Then the estimate update is:

$$\text{ARATE}_{\text{next}} = (1 - \text{ABETA}) * \text{ARATE} + \text{ABETA} * \frac{(\text{ANAZ} - \text{APOS})}{(\text{CTIME} - \text{ATIME})}$$

$$\text{APOS}_{\text{next}} = \text{ANAZ}$$

$$\text{ATIME}_{\text{next}} = \text{CTIME}$$

ABETA is a smoothing constant (approximately = .5). If ANAZ - APOS is negative, add 360°. Thus this difference is always taken as a positive angle. Check CTIME - ATIME. If too small (i.e., corresponds to less than 1/2 antenna sector) skip the update for this antenna sector.

2. Local Report Time of Measurement

The routine is used wherever local reports are utilized for track update or initialization. The input data are the measured report azimuth, AZR, and the antenna estimates.

The algorithm for measurement time, TM_{next} is:

$$TM_{next} = ATIME + \frac{(AZR - APOS)}{ARATE}$$

3. Coordinate Conversions

a. Local rho, theta to x, y.
b. Local x, y to rho, theta.
c. Remote rho, theta to local x, y.

See Section 4.2 and Figure 4-1 for a brief general description of the coordinate framework.

Note that remote sensor site parameters must be stored and used in c in the above list. A selection of parameters is made through the sensor ID supplied with each report.

4. Sector Thread Update

The requirements of this program (or programs) are to:

a. add a new track to a sector thread,
b. delete a track from a sector thread,
c. change a track from one sector thread to another.

5. ATCRBS/Radar Cross-reference (CREFA) Update

The requirements of this routine are to create or delete a CREFA link to a track State Vector.

6. DABS Cross-reference (CREFD) Update

The requirements of this routine are to create or delete a CREFD link to a track State Vector.

7. CREFA Reference

This routine locates a given track in CTS from its ATCRBS/radar surveillance file number by utilizing CREFA. Or it determines that no reference exists.

8. CREFD Reference

This routine provides a function similar to 7, but for a DABS ID using CREFD.

The following is a brief summary of required State Vector processing flags and indicators used in the Track and Report Processing Tasks. Their operation and function in the program are briefly summarized.

1. LOFL: local data flag
   RMFL: remote data flag

These flags indicate the source type of a report stored with a track in CTS. They are set when the report is stored and cleared during track update or initialization.

LOFL and RMFL are both used to indicate presence or absence of data from a particular site.

2. SMPR: smooth/predict flag
   SPRO: antenna sector process flag

These flags provide internal communication and prevent confusion in the timing of operations of the tracker.

SMPR is set when local reports are used for the Track Update or Track Initialization Process. The flag is cleared 1/2 scan after the Track Update Process has been performed for the aircraft's sector. Its function is to inform later program tasks that an update has already occurred on this scan and to defer further updates to the next scan.

SPRO is set after conclusion of local report processing in the Track Processing Task. It is cleared 1/2 scan after the Track Update Process has been performed for the aircraft's sector. It is tested before each track is accessed in the Track Processing Task. It inhibits reprocessing a track in the remote or late local report processing step.

3. DRSUR: drop surveillance flag
   DRATS: drop ATARS service flag

These flags indicate drop conditions. Both exist for the benefit of the State Vector Deletion Task, which takes final action when a track is to be dropped.

DRSUR is set by the tracker when it determines that a track should be dropped (drop bit received or too much time elapsed since last data input).

DRATS is set when the Track Update Process determines that the track is not qualified for ATARS service or is outside the service area. Otherwise, it is cleared during update.

4. ATSS: ATARS service flag

This flag is set by the Track Update Process when it determines that a track has become eligible for ATARS service. It adds a new track to the XINIT List. Geographical processing performs a more precise geographical check, and may clear this flag if the aircraft is outside the service area. The flag is cleared when the ATARS service is discontinued in response to a DRATS indication or because the geographic checks show the track to be outside the service area.

Three operations require coordinated actions between the Track Processing Task and subsequent ATARS tasks. These are to:

1. Start ATARS service for a track

2. Drop ATARS service for a track

3. Drop surveillance for a track

A surveillance drop is a total drop from the track and cross-reference files. An ATARS service drop only terminates traffic advisory and resolution advisory service; tracking is still required for domino processing.

These operations are coordinated by the State Vector flags; ATSS, DRATS, DRSUR. Table 4-7 shows in each case the actions initiated by track processing and then the actions taken by New Aircraft Processing or State Vector Deletion Tasks to complete the operation.

ATARS service is discontinued when a track leaves the ATARS service area. This event is determined by geographical processing. Geographical processing sets the DRATS flag directly and proceeds with the final actions as indicated in the table.

TABLE 4-7

STEPS REQUIRED TO START/DROP ATARS SERVICE OR DROP SURVEILLANCE

FUNCTION	STEP I	STEP II
1. Start ATARS Service	TRACK PROCESSING TASK Set ATSS Put track in XINIT List Reset DRATS	NEW AIRCRAFT PROCESSING TASK Remove from XINIT List Put in appropriate X-list Set INXFL Initialize State Vector
2. Drop ATARS Service	TRACK PROCESSING TASK Set DRATS Reset ATSS OR GEOGRAPHICAL PROCESSING Reset ATSS Set DRATS	
3. Drop ATARS/ Domino Surveillance	TRACK PROCESSING TASK Set DRSUR Erase CREFA link OR REPORT PROCESSING TASK Set DRSUR	STATE VECTOR DELETION TASK Remove track from appropriate X-list Remove from sector list and add to empties[1] Erase CREFD link Erase CREFA link

[1]This action effectively erases the track from CTS.

4.6 Pseudocode for Surveillance Report and Track Processing

The pseudocode follows for the Report Processing Task and the Track Processing Task described in this section. The format "surveillance data ITEM" used in the low level pseudocode refers to a data field in the DABS, ATCRBS or radar reports (Tables 3-3, 3-4, 3-5).

PSEUDOCODE TABLE OF CONTENTS

PSEUDOCODE TABLE OF CONTENTS

```
--------------------------------------------------------------------------------

   <*** THE PARAMETERS LISTED BELOW ARE LOCAL TO THE REPORT PROCESSING TASK ***>

STRUCTURE RPTPARM
  GROUP ztrk_init

     FLT ALT_TIME_FACT    < 7.0 >
     INT FIRMZR_INIT      < 5 >
     FLT ZVEL_INIT        < initial value of velocity for vertical tracker >

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

  <*** THE VARIABLES LISTED BELOW ARE LOCAL TO THE REPORT PROCESSING TASK ***>

STRUCTURE RPTVBL

<*** LOGIC-PATH VARIABLES ***>
  GROUP logic_path

     BIT RPTRK          < finished processing this report in report processing >

ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
------------------------------------------------------------------------------------

TASK REPORT_PROCESSING
   IN (one sectors reports in surveillance buffer, system parameters)
   OUT (RAR buffer, antenna sector list, deletion list, REMA, REMD, CREPD,
          CREPA, messages)
   INOUT (state vector, conflict tables, system varables);

     < Process all surveillance reports from sensor and do high level selection >
     Read sector header;
     Update antenna position and rate;

     REPEAT WHILE (more surveillance reports in sector need processing);

          CLEAR report finish flag;  <RPTRK>
          PERFORM ATC_drop_radar_reports;  <may set RPTRK>

          IF (report not finished yet)
               THEN PERFORM ATC_drop_non_mode_C_reports;  <may set RPTRK>
          IF (report not finished yet AND it is a remote report)
               THEN PERFORM remote_report_input;  <always sets RPTRK>
          Test if report is within ATARS/Domino surveillance area rho_theta mask;
          IF (report within surveillance mask)
               THEN SET report surveillance area mask flag;
          IF (report not finished yet AND (report in ATARS/domino
                    surveillance area OR null report))
               THEN PERFORM surveillance_area_report_processing:
                              <always sets RPTRK>

          IF (report not finished yet AND state vector exists)
               THEN Add aircraft to deletion list;
     ENDREPEAT;

END REPORT_PROCESSING;
```

----------------------- REPORT PROCESSING HIGH-LEVEL LOGIC ------------------------

```
---------------------------------------------------------------------------------

TASK REPORT_PROCESSING
   IN (one sectors reports in surveillance buffer, SYSTEM)
   OUT (RAR buffer, antenna sector list, deletion list, REMA, REMD, CREFD,
          CREFA, messages)
   INOUT (SVECT, conflict tables, SYSVAR);
     < Process all surveillance reports from sensor and do high level selection >

     Read sector header;
     Update antenna position (SYSVAR.APOS) and rate (SYSVAR.ARATE);

     REPEAT WHILE (more surveillance data for sector available);

          CLEAR RPTRK;
          PERFORM ATC_drop_radar_reports;  <may set RPTRK>

          IF (RPTRK EQ $FALSE)
               THEN PERFORM ATC_drop_non_mode_C_reports;  <may set RPTRK>
          IF (RPTRK EQ $FALSE AND surveillance data SENSOR ID NE
                                                  SYSTEM.LOCAL_ID)
               THEN PERFORM remote_report_input;  <always sets RPTRK>
          Test if report is within ATARS/Domino surveillance area
                    with rho_theta mask;
          IF (report within surveillance mask)
               THEN SET SVECT.SRVMSK;
          IF (RPTRK EQ $FALSE AND (SVECT.SRVMSK EQ $TRUE
                    OR surveillance data NULL REPORT EQ $TRUE))
               THEN PERFORM surveillance_area_report_processing;
                              <always sets RPTRK>

          IF (RPTRK EQ $FALSE AND SVECT NE null)
               THEN Add to deletion list;
     ENDREPEAT;

END REPORT_PROCESSING;

----------------------- REPORT PROCESSING LOW-LEVEL LOGIC ------------------------
```

```
--------------------------------------------------------------------------------
PROCESS ATC_drop_radar_reports;

     < Determine if radar report is not to be processed >

     IF ((ATC requests stop processing radar reports ) AND
               (this is a radar report))

          THEN IF (ATCRBS track initial flag set)  <ATIFLG>
                    THEN: <use this radar report since it is a one scan
                              substitution for ATCRBS surveillance report>
               ELSEIF (DABS ID or ATCRBS/radar surveillance file number in cross
                         reference)
                         THEN SET drop ATARS service and report finish flags;
                              Place aircraft on deletion list;
               OTHERWISE SET report finish flag;
          ELSE:  <continue processing report>

END ATC_drop_radar_reports;
```

```
--------------------------------------------------------------------------------
PROCESS ATC_drop_radar_reports;

     < Determine if radar report is not to be processed >

     IF ((SYSVAR.ATCROR EQ $FALSE) AND
                (surveillance data format type EQ radar report))

          THEN IF (SVECT.ATIFLG EQ $TRUE)
                    THEN: <radar report not rejected because of one scan
                              substitution for ATCRBS surveillance report>
               ELSEIF (DABS ID or ATCRBS/radar surveillance file number in cross
                         reference)
                         THEN SET SVECT.DRSUR and RPTRK;
                              Place on deletion list;
               OTHERWISE SET RPTRK;
          ELSE:  <continue processing report>

END ATC_drop_radar_reports;
```

```
------------------------------------------------------------------------------------

PROCESS ATC_drop_non_mode_C_report;

     < Determine if non_mode C report is to be processed >

     IF ((ATC requests stop processing of non_mode C reports ) AND
          (this is a non_mode C report) AND (non_mode C from last report))

          THEN IF (DABS ID or ATCRBS surveillance file number in cross reference)
               THEN SET drop ATARS service and report finish flags;
                    Place aircraft on deletion list;
               ELSE SET report finish flag;
          ELSE: <continue processing report>

END ATC_drop_non_mode_C_report;
```

```
--------------------------------------------------------------------------------

PROCESS ATC_drop_non_mode_C_report;

    < Determine if non_mode C report is to be processed >

    IF ((SYSVAR.ATCNMC EQ $FALSE) AND
        (surveillance data MODE C PRESENT EQ $FALSE) AND
        (SVECT.MCFLG = $FALSE))

        THEN IF (DABS ID or ATCRBS surveillance file number in cross reference)
                 THEN SET SVECT.DRSUR and RPTRK;
                      Place on deletion list;
                 ELSE SET RPTRK;
        ELSE:  <continue processing report>

END ATC_drop_non_mode_C_report;
```

```
PROCESS remote_report_input;

     < Special processing for remote reports >

     IF (null report)
          THEN SET report finish flag;
     ELSEIF (DABS ID or ATCRBS/radar surveillance file number in cross reference
               table)
          THEN Find track number;
               IF (drop ATARS service flag set)
                    THEN SET report finish flag;
                    ELSE Save report in state vector;
                         Calculate aircraft sector ID;
                         SET report finish flag;
                         IF (RAR message is empty AND aircraft is ATARS equipped
                                   AND aircraft is in ATARS service AND
                                   no resolution message in UPMES)
                              THEN Generate empty RAR report for this AC and
                                   send to own RAR buffer;
                         IF (report ATARS site ID bits have been changed
                              from last report)
                              THEN Store new ATARS site ID bits;
                                   IF (AC in a conflict table)
                                        THEN PERFORM seam_flag_updating;
     OTHERWISE SET report finish flag;

END remote_report_input;
```

```
-----------------------------------------------------------------------------
PROCESS remote_report_input;

     < Special processing for remote reports >

     IF (surveillance data NULL REPORT EQ $TRUE)
          THEN SET RPTRK;
     ELSEIF (DABS ID or ATCRBS/radar surveillance file number in cross reference
               table)
          THEN Find track number;
               IF (SVECT.DRSUR EQ $TRUE)
                    THEN SET RPTRK;
                    ELSE Save report in state vector;
                         SET SVECT.RMFL;
                         Calculate SVECT.SVSID;
                         SET RPTRK;
                         IF ((RAR message is empty) AND
                                   (SVECT.ATSEQ EQ $AEQ) AND
                                   (SVECT.ATSS EQ $TRUE AND
                                   (SVECT.UPHES EQ NULL))
                              THEN Generate empty RAR report for this AC and
                                   send to own RAR buffer;
                         IF (surveillance data UM FIELD <ATS SUBFIELD> NE SVECT.GEOG)
                              THEN SVECT.GEOG = surveillance data UM FIELD
                                                                      <ATS SUBFIELD>;
                                   IF (SVECT.CTPTR NE $NULL)
                                        THEN PERFORM seam_flag_updating;
     OTHERWISE SET RPTRK;

END remote_report_input;
```

```
--------------------------------------------------------------------------------
PROCESS surveillance_area_report_processing;

    < Final selection to determine if report is to be forwarded to track processing >

    IF (DABS ID or ATCRBS/radar surveillance file number in cross reference)

        THEN Find track number in cross reference;
            IF (drop ATARS service flag set) <DRSUR>
                THEN SET report finish flag;
            ELSEIF (track drop bit set)
                THEN SET drop ATARS service and report finish flags;
                    Place aircraft on deletion list;
            OTHERWISE PERFORM final_report_selection;  <may set RPTRK>

    ELSEIF (track drop bit set)
        THEN SET report finish flag;
    OTHERWISE PERFORM track_initialization;
            SET report finish flag;

END surveillance_area_report_processing;
```

```
--------------------------------------------------------------------------------

PROCESS surveillance_area_report_processing;

     < Final selection to determine if report is to be forwarded to track processing >

     IF (DABS ID or ATCRBS/radar surveillance file number in cross reference)

          THEN Find track number in cross reference;
               IF (SVECT.DRSUR EQ STRUE)
                    THEN SET RPTRK;
               ELSEIF (surveillance data TRACK DROP EQ STRUE)
                    THEN SET SVERT.DRSUR and RPTRK;
                         Place aircraft on deletion list;
               OTHERWISE PERFORM final_report_selection;  <may set RPTRK>

     ELSEIF (surveillance data TRACK DROP EQ STRUE)
          THEN SET RPTRK;
     OTHERWISE PERFORM track_initialization;
               SET RPTRK;

END surveillance_area_report_processing;
```

```
--------------------------------------------------------------------------------

PROCESS final_report_selection;

     IF (diffraction zone bit set OR null report)
          THEN SET report finish and null flags;
     ELSEIF (rho_theta test indicates that report is not reasonable)
          THEN SET report finish flag;
     OTHERWISE Store data in state vector;
               Calculate sector ID;
               SET local data and report finish flags;

               IF (RAR message is empty AND ATARS equipped AND
                         in ATARS service AND no resolution message in OPMES)
                    THEN Generate empty RAR report for this AC and send to own
                         RAR buffer;
               IF (site ID bits NE GEOG)
                    THEN Store new site ID bits;
                         IF (AC in a conflict table)
                              THEN PERFORM seam_flag_updating;
               IF (ALEC request from pilot)
                    THEN Create ALEC entry on designated AC PWILST;
                         Save ALEC update time in state vector;

END final_report_selection;
```

```
--------------------------------------------------------------------------------
PROCESS final_report_selection;

    IF ((surveillance data DIFFRACTION FLAG EQ $TRUE) OR
            (surveillance data NULL REPORT EQ $TRUE))
        THEN SET RPTRK and SVECT.NULLFG;
    ELSEIF (rho_theta test indicates that report is not reasonable)
        THEN SET RPTRK;
    OTHERWISE Store data in state vector;
              Calculate SVECT.SVSID;
              SET SVECT.LOFL and RPTRK;

              IF ((RAR message is empty) AND (SVECT.ATSEQ EQ $AEQ) AND
                      (SVECT.ATSS EQ $TRUE) AND (SVECT.UPHES EQ NULL))
                  THEN Generate empty RAR report for this AC and send to own
                      RAR buffer;
              IF (surveillance data UM FIELD <ATS SUBFIELD> NE SVECT.GEOG)
                  THEN SVECT.GEOG = surveillance data UM FIELD <ATS SUBFIELD>;
                      IF (SVECT.CTPTR NE $NULL)
                          THEN PERFORM seam_flag_updating;
              IF (surveillance data UM FIELD <AER SUBFIELD> EQ $TRUE)
                  THEN Create ALEC entry on designated AC PWILST;
                      SVECT.ALECT = SYSVAR.CTIME;

END final_report_selection;
```

```
--------------------------------------------------------------------------------

PROCESS flag_pointer_initial

     Initialize various flags;

     Initialize pointers;

     Initialize equipage;

END flag_pointer_initial;
```

```
-------------------------------------------------------------------------------
PROCESS flag_pointer_initial;

     SET SVECT.SMPR;
     CLEAR SVECT.ATSS;
     CLEAR SVECT.DRSUR;
     CLEAR SVECT.DRATS;
     CLEAR SVECT.LOFL;
     CLEAR SVECT.RMFL;
     SVECT.CTE = $NULL;
     SVECT.CTPTR = $NULL;
     SVECT.CUNC = surveillance data TARGET CONTROL STATE;
     SVECT.ATSEQ = $UNEQ;

END flag_pointer_initial;
```

```
PROCESS seam_flag_updating;

    < Determine state of seam flag in conflict table >

    Find all sites that see any aircraft in conflict;
    Determine whether any of these sites are connected;

    IF (no sites are connected)
        THEN CLEAR seam flag in conflict table;
        ELSE SET seam flag;

END seam_flag_updating;
```

```
--------------------------------------------------------------------------------------
PROCESS seam_flag_updating;

     < Determine state of seam flag in conflict table >

     Form logical OR of GEOG fields for all AC in conflict table;
     Form logical AND between above result and connected site data;

     IF (result is zero)
          THEN CLEAR CTHEAD.SEAM;
          ELSE SET CTHEAD.SEAM;

END seam_flag_updating;
```

```
--------------------------------------------------------------------------------

PROCESS track_initialization;

     Find empty track slot;
     Store report time;
     Convert report rho, theta to x, y, z coordinates;
     Set up state vector initial conditions;
     Add aircraft to proper antenna sector list;
     Calculate proper ATARS sector ID for aircraft;

     IF (mode C altitude present)
          THEN PERFORM ztrack_init;
               Initialize vertical firmness;
               SET valid altitude report flag;  <MCFLG>
          ELSE Zero vertical firmness;
               CLEAR valid altitude report flag;  <MCFLG>

     PERFORM flag_pointer_initial;
     Clear REMA or REMD if report duplicate;

     IF (DABS report)
          THEN Link to CREFD and set type to DABS;
               Save DABS ID;
               Generate messages to report climb performance capability, equipage,
                    and class of service;
     ELSEIF (radar report)
          THEN Link to CREFA;
     OTHERWISE SET ATCRBS initialization flag and set type to ATCRBS;
               Save ATCRBS ID;
               Add ATCRBS file number to state vector;
               Link to CREFA;

END track_initialization;
```

```
--------------------------------------------------------------------------------------
PROCESS track_initialization;

     Find empty track slot;
     SVECT.TM = SYSVAR.CTIME;
     Convert report rho, theta to x, y, z coordinates;
     SVECT.FIRME = 1;
     SVECT.FIRMI = 1;
     SVECT.RMS = 0.;
     Add aircraft to proper antenna sector list;
     Calculate SVECT.SVSID;

     IF (surveillance data MODE C PRESENT EQ $TRUE)
          THEN PERFORM ztrack_init;
               SVECT.FIRMZ = 1;
               SET SVECT.MCFLG;
          ELSE SVECT.FIRMZ = 0;
               CLEAR SVECT.MCFLG;
     PERFORM flag_pointer_initial;
     Clear REMA or REMD if report duplicate;

     IF (surveillance report format type EQ DABS)
          THEN Link to CREFD;
               SVECT.TYPE = $DABS;
               SVECT.CODE = surveillance data DABS ID;
               Send DATA LINK CAPABILITY REQUEST and ATARS EXTENDED CAPABILITY
                    REQUEST messages;
     ELSEIF (surveillance report format type EQ radar)
          THEN Link to CREFA;
     OTHERWISE SET SVECT.ATIFLG;
               SVECT.TYPE = $ATCRBS;
               SVECT.CODE = surveillance data ATCRBS SURVEILLANCE FILE NO;
               Link to CREFA;
END track_initialization;


---------------------- REPORT PROCESSING LOW-LEVEL LOGIC ------------------------
```

```
--------------------------------------------------------------------------------

PROCESS ztrack_init;

     < Initialize vertical tracker quantities >

     Initialize z tracker state vector data;

END ztrack_init;
```

```
--------------------------------------------------------------------------------
PROCESS ztrack_init;

     < Initialize vertical tracker quantities >

     SVECT.ZS = surveillance data MODE C ALTITUDE;
     SVECT.ZDE = ZVEL_INIT;
     SVECT.TMZ = SYSVAR.CTIME;
     SVECT.ZPREV = surveillance data MODE C ALTITUDE;
     SVECT.TLUPD = SYSVAR.CTIME - SYSTEM.DT;
     SVECT.LOT = SYSTEM.Q / ABS(SVECT.ZDE);
     SVECT.FIRMZR = FIRMZR_INIT;
     SVECT.SUCNT = 0;
     SVECT.SUMRES = 0.;
     SVECT.TTPRAL = SYSVAR.CTIME - SVECT.LOT + ALT_TIME_FACT * SYSTEM.DT;

END seam_flag_updating;
```

```
--------------------------------------------------------------------------------

    <*** THE PARAMETERS LISTED BELOW ARE LOCAL TO THE TRACK PROCESSING TASK ***>


STRUCTURE TRKPARM


  GROUP vert_tracker
     FLT BETA_MAX          < 0.1 >
     FLT CNT_DELT          < 4.0 >
     INT CNT_INCR          < 10 >
     FLT DECAY_FCTR        < decay of transition >
     FLT DISCREPANCY       < triggers rate reinitialization >
     FLT FIRMZ_MAX         < 9.0 >
     FLT FIRMZR_INCR       < 0.6 >
     FLT FIRMZR_MAX        < 10.0 >
     FLT FIRMZR_MIN        < 2.0 >
     FLT LEVEL_TIME        < 99.0 >
     FLT LIL_BIT           < 1.0 >
     FLT LOT_SCALE         < 0.4 >
     FLT MISS_FCTR         < for missing data >
     FLT ONEXRATE          < single transition rate >
     FLT PART_SCAN         < 0.8 >
     FLT RATE_FACT         < 2.0 >
     FLT RATESMOOTH        < stiff rate smoothing >
     FLT SCAN_FACTOR       < 0.05 >
     INT SHIFT_FACT        < 64 >
     FLT SR_MAG            < magnitude setting >
     INT SR_THRESH         < detect excess summed residual >
     FLT SRGAIN            < summed residual smoothing gain >
```

PRECEDING PAGE BLANK-NOT FILMED

```
------------------------------------------------------------------------------------

    FLT TEST_THRSH       < 100.0 ft >

    FLT TRANS_FACTOR     < 1.2 >

    FLT XLEVEL           < transition level times >

    FLT XTONORM          < threshold for transition to normal smoothing >

    FLT XTRARESID        < for extra summed residual >

    FLT ZCORRECT         < for correcting altitude rate >

    FLT ZSMOOTH          < position smoothing >


  GROUP trk_quality

    FLT FESTAB           < firmness value used to determine if track is qualified
                           for ATARS processing >

    FLT TDOS             < scan time correct threshold >

    FLT TDROP            < time interval without horizontal reply to drop a track >

    FLT THMS             < time required to zero horizontal maneuver status >


ENDSTRUCTURE;
```

```
-----------------------------------------------------------------------------------

    <*** THE VARIABLES LISTED BELOW ARE LOCAL TO THE TRACK PROCESSING TASK ***>

STRUCTURE TRKVBL

  GROUP vert_tracker

     FLT BETA1            < altitude rate smoothing >
     FLT BLIM             < TRK_VAR.BETA1 limit >
     INT DBINS            < size of level occupancy bins >
     FLT DELT             < change in level occupancy time >
     FLT DZM              < change in MODE C report >
     FLT DZ10             < scan fraction of TRK_VAR.DELT >
     INT ISGN             < sign applied to SYSVAR.Q >
     FLT QSIGN            < sign of bin quantization >
     FLT TCUR             < current transition time >
     FLT TEST             < test for altitude trend >
     FLT TNDEX            < transition index >
     FLT TPREV            < previous transition time >
     FLT ZR               < surveillance data MODE C ALTITUDE >
     FLT Z7               < previous level occupancy time >

  GROUP logic_path

     BIT HIT              < report is processed as a 'hit' or a good report >
     BIT RRREJF           < remote report processing finished in track
                             processing task >
     BIT TSREJF           < track service finish flag >
```

```
------------------------------------------------------------------------------

 GROUP smoothing

    FLT D2              < cross track distance >
    FLT D2TH            < threshold for turn sense >
    INT IHIT            < turn sense weight >
    FLT NEW_HMS         < new turn sense >
    FLT S               < horizontal maneuver status >
    FLT TH1             < lower cross track deviation threshold >
    FLT TH2             < upper cross track deviation threshold >
    FLT TURN            < turn sense state >
    FLT W               < turn rate >
    FLT XA              < alpha smoothed x position >
    FLT XDEN            < new external x velocity >
    FLT XDIN            < new internal x velocity >
    FLT XS              < external smoothed x position >
    FLT XSI             < internal smoothed x position >
    FLT YA              < alpha smoothed y position >
    FLT YDEN            < new external y velocity >
    FLT YDIN            < new internal y velocity >
    FLT YS              < external smoothed y position >
    FLT YSI             < internal smoothed y position >

 GROUP predict

    FLT A               < time to next data report >
    FLT DS              < scan time of antenna to next local report >
    FLT NEW_TM          < new computed time of track >
    BIT RECFLG          < recalculate prediction >
    FLT TDDS            < scan time correct threshold >
    FLT THMS            < time required to zero horzontal maneuver status >

ENDSTRUCTURE;

----------------------- TRACK PROCESSING LOCAL VARIABLES -----------------------
```

```
--------------------------------------------------------------------------------------------
TASK TRACK_PROCESSING
   IN (antenna sector list, system variables, system parameters)
   OUT (deletion list, x/ex_list, messages, CREFA, CREFD, XINIT list)
   INOUT (state vector);
     < Takes reports passed from report processing and initializes or updates tracks >
     REPEAT WHILE (more tracks on antenna sector list from four sectors previous
                   to the present processing sector)
          IF (smooth_predict OR antenna sector process flag set)
               THEN: <proceed to next track>
          ELSEIF (null report)
               THEN CLEAR null flag;
                    SET antenna sector process flag;  <SPRO>
                    CLEAR hit flag;  <HIT>
                    PERFORM track_update;  <for miss>
          OTHERWISE SET hit flag;  <HIT>
                    PERFORM track_update;  <for hit>
                    SET antenna sector process flag;  <SPRO>
     ENDREPEAT;
     REPEAT WHILE (more track data in antenna sector list not from four sectors
                   previous to the present processing sector)
                   <remote hit or late local>
          IF (smooth_predict flag set)
               THEN:
          ELSEIF (antenna sector process flag not set)
               THEN SET hit flag;  <HIT>
                    PERFORM track_update;  <for hit>
          OTHERWISE:  <continue to next track>
          CLEAR smooth_predict and antenna sector process flags;
     ENDREPEAT;
     REPEAT WHILE (more tracks remain on back side antenna sector list)
          Select next state vector from list;
          CLEAR smooth_predict and antenna sector process flags;
     ENDREPEAT;
END track_processing;

----------------------  TRACK PROCESSING HIGH-LEVEL LOGIC  ------------------------
```

```
--------------------------------------------------------------------------------

TASK TRACK PROCESSING
   IN (antenna sector list, SYSVAR, SYSTEM)
   OUT (deletion list, x/ex_list, messages, CREFA, CREFD, XINIT list)
   INOUT (SVECT);
     < Takes reports passed from report processing and initializes or updates tracks >
     REPEAT WHILE (more tracks on antenna sector list from four sectors previous
                   to the present processing sector)
          IF ((SVECT.SHPR EQ $TRUE) OR (SVECT.SPRO EQ $TRUE))
               THEN: <proceed to next track>
          ELSEIF (SVECT.NULLFG EQ $TRUE)
               THEN CLEAR SVECT.NULLFG;
                    SET SVECT.SPRO;
                    CLEAR HIT;
                    PERFORM track_update:  <for miss>
          OTHERWISE SET HIT;
                    PERFORM track_update:  <for hit>
                    SET SVECT.SPRO;
     ENDREPEAT;
     REPEAT WHILE (more track data in antenna sector list not from four sectors
                   previous to the present processing sector)
                   <remote hit or local>
          IF (SVECT.SHPR EQ $TRUE)
               THEN;
          ELSEIF (SVECT.SPRO EQ $FALSE)
               THEN SET HIT;
                    PERFORM track_update: <for hit>
          OTHERWISE:  <continue to next track>
          CLEAR SVECT.SHPR and SVECT.SPRO;
     ENDREPEAT;
     REPEAT WHILE (more tracks remain on back side antenna sector list)
          Select next state vector from list;
          CLEAR SVECT.SPRO and SVECT.SHPP;
     ENDREPEAT;
END track_processing;

------------------------ TRACK PROCESSING LOW-LEVEL LOGIC ------------------------
```

```
--------------------------------------------------------------------------------

PROCESS track_update;

     < Smooth and predict track of aircraft >

     CLEAR RRREJF;
     IF (a hit track update)
          THEN Initialize data;
               Correct prediction and local slant range to current time:
               Convert rho, theta to local x, y coordinates;

               IF (remote report)
                    THEN PERFORM remote_report_service;  <may set RRREJF>
                    ELSE PERFORM local_report_service;
               IF (remote report finish flag not set)
                    THEN PERFORM x_y_smoothing;
                         PERFORM z_tracker;
                         Save time, control status;
                         CLEAR remote and local flags;

          ELSE SET local sensor lost data link contact flag;  <DLOUT>
               IF (too long since last report)
                    THEN SET drop ATARS surveillance flag;  <DRSUR>
                         Place aircraft on deletion list;
                         Unlink aircraft from x/ex_list;
                    ELSE PERFORM z_tracker;

     IF (remote report finish flag not set)
          THEN PERFORM track_service_determination;

END track_update;
```

```
PROCESS track_update;

     < Smooth and predict track of aircraft >

     CLEAR RRREJF;

     IF (HIT = $TRUE)
          THEN Compute NEW_TM;
               Correct prediction and local slant range to current time;
               Convert rho, theta to local x, y coordinates;

               IF (SVECT.RMFL EQ $TRUE)
                    THEN PERFORM remote_report_service;  <may set RRREJF>
                    ELSE PERFORM local_report_service;
               IF (RRREJF EQ $FALSE)
                    THEN PERFORM x_y_smoothing;
                         PERFORM z_tracker;
                         SVECT.TM = NEW_TM;
                         SVECT.CUNC = surveillance data TARGET CONTROL STATE;
                         CLEAR SVECT.RMFL and SVECT.LOFL;

          ELSE SET SVECT.DLOUT;
               IF ((SYSVAR.CTIME - SVECT.TM) GT TDROP)
                    THEN SET SVECT.DRSUR;
                         Place aircraft on deletion list;
                         Unlink aircraft from x/ex_list;
                    ELSE PERFORM z_tracker;

     IF (RRREJF EQ $FALSE)
          THEN PERFORM track_service_determination;

END track_update;
```

```
PROCESS altitude_transition_logic;

     Determine sign of mode C report change;
     Assign that sign to the bin quantization;
     Estimate altitude trend;

     IF (altitude trend test fails)
          THEN PERFORM single_transition;
               PERFORM level_transition_update;
          ELSE Calculate change in level occupancy time;
               IF (estimated level occupancy time LT scan length)
                    THEN Reassess change in level occupancy time;

               IF (vertical rate firmnes indicates level track) OR
                    (discrepancy in bin occupancy time is beyond its limit)
                    THEN PERFORM rate_reinitialization;

                    ELSE PERFORM residual_rate_detection;
                         Smooth estimated time, altitude, and altitude rate;
               PERFORM level_transition_update;

END altitude_transition_logic;
```

```
-----------------------------------------------------------------------------------

PROCESS altitude_transition_logic;

     ISGN = INT( SIGN( 1., DZM));
     QSIGN = SYSTEM.Q * ISGN;
     TEST = SVECT.ZDE * DZM;

     IF (TEST LE TEST_THRSH AND DBINS EQ 1)
          THEN PERFORM single_transition;
               PERFORM level_transition_update;
          ELSE TPREV = (SYSVAR.CTIME - SVECT.TTPRAL) / DBINS;
               DELT = TPREV - SVECT.LOT;
               DZ10 = DELT / SYSTEM.DT;
               IF (SVECT.LOT LT SYSTEM.DT)
                    THEN DZ10 = ((SYSVAR.CTIME - SVECT.TLUPD) / SVECT.LOT) - DBINS;

               IF ((SVECT.FIRMZR LE 0) OR (ABS( DZ10) GT DISCREPANCY))
                    THEN PERFORM rate_reinitialization;
                         PERFORM level_transition_update;

                    ELSE PERFORM residual_rate_detection;
                         SVECT.LOT = (SVECT.LOT + BETA1 * < DBINS * TPREV
                                     -SVECT.LOT>) / (1. + BETA1 * <DBINS - 1>);
                         SVECT.ZDE = QSIGN / SVECT.LOT;
                         SVECT.ZS = SVECT.ZP + ZSMOOTH * (ZR _ SVECT.ZP);

               PERFORM level_transition_update;

END altitude_transition_logic;
```

```
--------------------------------------------------------------------------------

PROCESS level_transition_update;

     Update previously reported mode C altitude;
     Update time of transition to previously reported mode C altitude;
     Update startup counter;

     IF (last altitude report was received before last track update)
          THEN correct transition to fall within period of missing data;
          ELSE;

END level_transition_update;
```

```
--------------------------------------------------------------------------------
PROCESS level_transition_update;

     SVECT.ZPREV = ZR;
     SVECT.TTPRAL = SYSVAR.CTIME;
     SVECT.SUCNT = SVECT.SUCNT + CNT_INCR;

     IF (SVECT.TMZ LT SVECT.TLUPD)
          THEN SVECT.TTPRAL = SYSVAR.CTIME + MISS_FCTR * ( SVECT.TMZ - SYSVAR.CTIME
                                                       + SYSTEM.DT) ;
          ELSE;

END level_transition_update;
```

```
---------------------------------------------------------------------------------

PROCESS local_report_service;

     SET smooth_predict flag;  <SMPP>

     IF (not DABS report)
          THEN; <proceed to smoothing>
     ELSEIF (remote RARs being received)
          THEN Send message to remote sensor to stop sending RAR data;
               Place negative remote ATARS site ID into REMRAR;
               Send message to own DABS sensor to start ATARS service for
                    this AC;
               CLEAR local sensor lost data link contact flag;  <DLOUT>
     OTHERWISE CLEAR local sensor lost data link contact flag;  <DLOUT>
                    <the remote site ID in REMRAR is where data may be requested>

END local_report_service;
```

```
--------------------------------------------------------------------------------------
PROCESS local_report_service;

     SET SVECT.SMPR;

     IF (SVECT.TYPE NE $DABS)
          THEN: <proceed to smoothing>
     ELSEIF (SVECT.REMRAR GT 0)
          THEN Send START/STOP REMOTE RAR DATA message to site in SVECT.REMRAR,
               specifying STOP;
               SVECT.REMRAR = - SVECT.REMRAR;
               Send START ATARS SERVICE message for this AC to own sensor;
               CLEAR SVECT.DLOUT;
     OTHERWISE CLEAR SVECT.DLOUT;
                    <the remote site ID in REMRAR is where data may be requested>

END local_report_service;
```

```
--------------------------------------------------------------------------------

PROCESS nontransition_logic;

     Evaluate smoothed altitude;

     Determine current level occupancy time;

     Create level occupancy time index;


     IF (estimated level occupancy time LT 80% of a scan)
          THEN Reassign level occupancy time index;


     IF (level occupancy time index indicates level flight)
          THEN Transition to level flight;
     ELSEIF (level occupancy time index indicates excess altitude rate)
          THEN Alter external altitude rate;
               Decrease altitude rate firmness;
     ELSEIF (altitude rate firmness is not reinforced)
          THEN Allow altitude rate to decay normally;
     OTHERWISE;

END nontransition_logic;
```

```
-------------------------------------------------------------------------------
PROCESS nontransition_logic;

     SVECT.ZS = SVECT.ZP + ZSMOOTH * ( ZR - SVECT.ZP);
     TCUR = SYSVAR.CTIME - SVECT.TTPRAL + SYSTEM.DT;
     TNDEX = (TCUR - SVECT.LOT) / SYSTEM.DT;

     IF (SVECT.LOT LT <PART_SCAN * SYSTEM.DT>)
          THEN TNDEX = TCUR / SVECT.LOT;

     IF (TNDEX GT XLEVEL)
          THEN SVECT.ZS = ZR;
               SVECT.ZDE = 0.;
               SVECT.LOT = LEVEL_TIME;
               SVECT.FIRMZR = 0.;
               SVECT.SUMRES = 0.;
     ELSEIF (TNDEX GE ZCORRECT)
          THEN SVECT.ZDE = SIGN( SYSTEM.Q, SVECT.ZDE) / ( SVECT.LOT +
                             (LOT_SCALE * SVECT.LOT + SYSTEM.DT) *
                             (TNDEX - LOT_SCALE)**2 );
               SVECT.FIRMZR = MAX( FIRMZR_MIN, <SVECT.FIRMZR - 1.>);
     ELSEIF (SVECT.FIRMZR LT 1.)
          THEN SVECT.ZDE = SVECT.ZDE * DECAY_FCTR;
               SVECT.LOT = SYSTEM.Q / (ABS ( SVECT.ZDE ) + LIL_BIT);
     OTHERWISE;

END nontransition_logic;
```

```
---------------------------------------------------------------------------------------

PROCESS rate_reinitialization;

     Save estimated level occupancy time;
     Calculate level occupancy time for level track;
     Estimate altitude rate;
     Set summed residual to zero;
     Set rate firmness to indicate a single transition;
     Calculate smoothed altitude;

END rate_reinitialization;
```

```
--------------------------------------------------------------------------------

PROCESS rate_reinitialization;

     SVECT.LOT = TRANS_FACTOR * TPREV + SCAN_FACTOR * SYSTEM.DT;

     SVECT.ZDE = QSIGN / SVECT.LOT;

     SVECT.SUMRES = 0.;

     SVECT.FIRMZR = 1.;

     SVECT.ZS = ZR - ( QSIGN / RATE_FACT ) + SVECT.ZDE * (SYSTEM.DT / RATE_FACT);

END rate_reinitialization;
```

```
---------------------------------------------------------------------------
PROCESS remote_report_service;

     < Determine if remote report service is needed for a report >

     IF (report is not acceptable according to x, y reasonableness test)
          THEN SET remote report finish flag:  <RRREJF>
     ELSEIF (RAR remote site ID set to zero)  <REMRAR>
          THEN Place negative remote site ID into REMRAR;
     OTHERWISE: <use present RAR remote site ID >

     IF ((remote report finish flag not set) AND
          (cone of silence flag set) AND
          (ATARS service flag set) AND
          (AC is ATARS equipped) AND
          (remote ATARS site identification from which data will be requested
               is set))
          THEN Send message to remote site requesting RAR data;
               Place absolute value of remote ATARS site ID in REMRAR;
               Send message to own DABS sensor to stop ATARS service for
                    this AC;
          ELSE:  <continue processing remote report>

END remote_report_service;
```

```
PROCESS remote_report_service;

     < Determine if remote report service is needed for a report >

     IF (report is not acceptable according to x, y reasonableness test)
          THEN SET RRREJF;
     ELSEIF (SVECT.REMRAR = 0)
          THEN SVECT.REMRAR = - ( surveillance data SENSOR ID );
     OTHERWISE; <use present RAR remote site ID >

     IF ((RRREJF EQ $FALSE) AND
          (surveillance data ZENITH CONE FLAG EQ $TRUE) AND
          (SVECT.ATSS EQ $TRUE) AND
          (SVECT.ATSEQ NE SUNEQ) AND
          (SVECT.REMRAR LT 0)
          THEN Send START/STOP REMOTE RAR DATA message to site indicated
                in SVECT.REMRAR, specifying START and one-scan flag not set;
          THEN SVECT.REMRAR = ABS( SVECT.REMRAR );
               Send STOP ATARS SERVICE MESSAGE for this AC to own sensor;
          ELSE:  <continue processing remote report>

END remote_report_service;
```

```
PROCESS residual_rate_detection;

     Determine summed residual;

     IF (excessive summed residual is detected)
          THEN Assign altitude rate smoothing parameter;
               Reset altitude rate firmness;
               Reset summed residual;
          ELSE Assign altitude rate smoothing parameter;
               Correct altitude rate firmness;

END residual_rate_detection;
```

```
--------------------------------------------------------------------------------
PROCESS residual_rate_detection;

     SVECT.SUMRES = SRGAIN * SVECT.SUMRES + DZ10;

     IF (ABS( SVECT.SUMRES) GT SR_THRESH)
          THEN BETA1 = XTRARESID;
               SVECT.FIRMZR = FIRMZR_MIN;
               SVECT.SUMRES = SIGN( SR_MAG, SVECT.SUMRES);
          ELSE Z7 = SVECT.LOT;
               BLIM = (Z7 - 1.)**2 / (Z7**2 + SHIFT_FACT);
               BETA1 = MAX(<1. / (SVECT.FIRMZR + FIRMZR_INCR)>, BLIM, BETA_MAX);
               SVECT.FIRMZR = MIN(SVECT.FIRMZR + 1., FIRMZR_MAX);

END residual_rate_detection;
```

```
--------------------------------------------------------------------------------------

PROCESS single_transition;

     < Assign variables for a single observed transition. >

     Reset external estimated altitude rate;
     Reset external smoothed altitude;
     Reset internal estimated level occupancy time;
     Reset altitude rate firmness;
     Reset summed residual;

END single_transition;
```

```
-------------------------------------------------------------------------------

PROCESS single_transition;

     < Assign variables for a single observed transition. >

     SVECT.ZDE = ONEXRATE * ISGN;
     SVECT.ZS = ZR - ( QSIGN / RATE_FACT ) + SVECT.ZDE * ( SYSTEM.DT / RATE_FACT);
     SVECT.LOT = QSIGN / SVECT.ZDE;
     SVECT.FIRHZR =0;
     SVECT.SUMRES = 0.;

END single_transition;
```

```
----------------------------------------------------------------------------------
PROCESS startup_smoothing;

     Smooth altitude;
     Smooth altitude rate;

     IF (change in mode C report is detected)
          THEN PERFORM level_transition_update;
          ELSE;

END startup_smoothing;
```

```
--------------------------------------------------------------------------------

PROCESS startup_smoothing;

     SVECT.ZS = SVECT.ZP + ZSMOOTH *(ZR - SVECT.ZP);
     SVECT.ZDE = SVECT.ZDE + RATESMOOTH * (( ZR - SVECT.ZP) / (SYSVAR.CTIME
          - SVECT.TLUPD));

     IF (DZM NE 0.)
          THEN PERFORM level_transition_update;
          ELSE;

END startup_smoothing;
```

```
--------------------------------------------------------------------------------

PROCESS track_service_determination;

     CLEAR track service finish flag;  <TSREJF>
     IF (drop surveillance flag not set)
          THEN PERFORM x_y_prediction;
     ELSEIF (AC not on x/ex_list)
          THEN Drop track and erase CREFA or CREFD link;
               SET track service finish flag;
     OTHERWISE IF (track ATCRBS or radar only)
          THEN Erase CREFA link;
     IF (trk service finish flag not set and vert firmness zero)
          THEN CLEAR altitude flag;  <NCFLG>
     IF (track service finish flag not set AND level firmness test)
          THEN IF (AC not on x/ex_list)
                    THEN Add AC to xinit list;
               IF (ATARS service flag set)  <ATSS>
                    THEN CLEAR drop ATARS service flag;  <DRATS>
               ELSEIF (track within service mask from x, y masking)
                    THEN SET ATARS service flag;  <ATSS>
                         CLEAR drop ATARS service flag;  <DRATS>
               OTHERWISE;  <continue service>
          ELSE IF (AC on x/ex_list)
               THEN SET drop ATARS service flag;  <DRATS>
                    Place aircraft on deletion list;
                    Unlink from x/ex_list;
                    CLEAR ATARS service flag;  <ATSS>
     IF (track service finish flag not set AND DABS track)
          THEN Update primary_secondary status in state vector;  <PSTAT>

END track_service_determination;
```

```
--------------------------------------------------------------------------------

PROCESS track_service_determination;

     CLEAR TSREJF;
     IF (SVECT.DRSUR EQ $FALSE)
          THEN PERFORM x_y_prediction;
     ELSEIF (SVECT.NEXTX and SVECT.PREVX EQ $NULL)
          THEN Drop track and erase CREFA or CREFD link;
               SET TSREJF;
     OTHERWISE IF (SVECT.TYPE NE $DABS)
          THEN Erase CREFA link;
     IF (TSREJF EQ $FALSE AND SVECT.FIRMZ EQ 0)
          THEN CLEAR SVECT.MCFLG;
     IF (TSREJF EQ $FALSE AND SVECT.FIRMI GE FESTAB)
          THEN IF (SVECT.NEXTX and SVECT.PREVX EQ $NULL)
                    THEN Add AC to XINIT list;
               IF (SVECT.ATSS EQ $TRUE)
                    THEN CLEAR SVECT.DRATS;
               ELSEIF (SVECT.SRVMSK EQ $TRUE)
                    THEN SET SVECT.ATSS;
                         CLEAR SVECT.DRATS;
               OTHERWISE: <continue service>
          ELSE IF (SVECT.NEXTX or SVECT.PREVX NE $NULL)
               THEN SET SVECT.DRATS;
                    Place aircraft on deletion list;
                    Unlink from x/ex_list;
                    CLEAR SVECT.ATSS;
     IF ((TSREJF EQ $FALSE) AND (SVECT.TYPE EQ $DABS))
          THEN SVECT.PSTAT = surveillance data SENSOR PRIORITY STATUS;

END track_service_determination;
```

```
--------------------------------------------------------------------------------------

PROCESS x_y_prediction;

     < Compute predicted values for track >

     CLEAR recalculate flag;  <RECFLG>
     Estimate scan time of antenna to next local report;  <DS>
     IF (not hit report)
          THEN Insert internal predicted pos, vel and time into stack;
               Place previous pred (pos & vel) into new smoothed values;
     ELSEIF (not remote data)
          THEN;
     OTHERWISE IF (first pass through prediction for remote data)
          THEN Compute scan time;
          ELSE Compute scan time;
               SET recalculate flag;  <RECFLG>
     LOOP;
          Compute internal and external pred x, y pos, range, & azimuth;
          Compute scan time correction and stack time;
          Compute XPINEW, YPINEW;
     EXITIF (recalculate flag set);
               IF (abs value extra scan time GE scan time correct threshold)
                    THEN Recompute scan time;  <DS>
                         SET recalculate flag;  <RECFLG>
     ENDLOOP;
     IF (miss report)
          THEN Calculate time to next data;  <A>
               IF (time to next data GT time required to zero horizontal
                         maneuver status)
                    THEN Zero horizontal maneuver status;
     Compute next data time;  <TNP>
     Save range, azimuth, next data time;

END x_y_prediction;


----------------------- TRACK PROCESSING HIGH-LEVEL LOGIC -------------------------
```

```
--------------------------------------------------------------------------------------

PROCESS x_y_prediction;

     < Compute predicted values for track >

     CLEAR RECFLG;
     Compute DS;
     IF (HIT = $FALSE)
          THEN Insert SVECT.XPI, SVECT.YPI, SVECT.XDI, SVECT.YDI, SVECT.TH into stack;
               Place SVECT.XP, SVECT.YP into XS, YS;
               Place SVECT.XDE, SVECT.YDE into SVECT.XDI, SVECT.YDI;
     ELSEIF (SVECT.RMPL EQ $FALSE)
          THEN;
     OTHERWISE IF (first pass through prediction for remote data)
          THEN Compute DS;
          ELSE Compute DS;
               SET RECFLG;
     LOOP;
          Compute internal and external pred x, y pos, range, & azimuth;
          Compute scan time correction and stack time;
          Compute XPINEW, YPINEW;
     EXITIF (RECFLG EQ $TRUE);
               IF (ABS( DDS ) GE TDDS)
                    THEN Recompute DS;
                         SET RECFLG;
     ENDLOOP;
     IF (HIT = $FALSE)
          THEN A = SVECT.TNP - SVECT.TH;
               IF (A GT THMS)
                    THEN SVECT.HMS = 0.;
     Compute SVECT.TNP;
     Save SVECT.RHOP, SVECT.AZP, SVECT.TNP;

END x_y_prediction;
```

```
-----------------------------------------------------------------------------

PROCESS x_y_smoothing;

     Save old velocity estimate and predicted position;
     Determine horizontal maneuver status, cross track distance,
          and threshold for turn sense;
     Compute turn rate;
     IF (cross track dist LE threshold for turn sense) <D2 LE D2TH>
          THEN Zero turn sense;  <HMS(next)=0>
               Compute internal velocity components; <XDI(next), YDI(next)>
               Place internal velocity into external velocity;
          ELSE Compute internal velocity components modified by half
                    angle correction;  <XDI(next),YDI(next)>
               IF (present horiz maneuver status EQ old status)
                    THEN Compute ext vel components;  <XDE(next), YDE(next)>
                         Repeat turn sense;  <HMS(next) = HMS>
                    ELSE Turn sense changed;  <HMS(next) = S>
                         Place internal velocity into external velocity;
     Place alpha smoothed position into internal position:  <XSI,YSI>
     Place new position, velocity, and time in stack;
     Compute smoothed external position <XS, YS>, lower and upper
          cross track deviation threshold;  <TH1, TH2>
     IF (cross track distance GT lower threshold)
          THEN IF (cross track distance GT upper threshold)
                    THEN Turn sense weight is two;
                    ELSE Turn sense weight is one;
               Compute turn sense direction; <IHIT = IHIT * S>
          ELSE Zero turn sense weight;
     Determine turn sense state;
     Update internal and external firmness level;

END x_y_smoothing;
```

```
-----------------------------------------------------------------------------------

PROCESS x_y_smoothing;

     Save old velocity estimate and predicted position;
     Determine S, D2, and D2TH;
     Compute W;
     IF (D2 LE D2TH)
          THEN NEW_HMS = 0.;
               Compute XDIN, YDIN;
               Place XDIN, YDIN into XDEN, YDEN;
          ELSE Compute XDIN, YDIN modified by half angle correction;
               IF (S EQ SVECT.HMS)
                    THEN Compute XDEN, YDEN;
                              < present turn sense is okay >
                    ELSE SVECT.HMS = S;
                         Place XDIN, YDIN into XDEN, YDEN;
     Place XA, YA into XSI, YSI;
     Place new internal position, velocity, and time in stack;
     Compute XS, YS and TH1, TH2;
     IF (D2 GT TH1)
          THEN IF (D2 GT TH2)
                    THEN IHIT = 2;
                    ELSE IHIT = 1;
               IHIT = IHIT * S;
          ELSE IHIT = 0;
     Determine TURN;
     Update SVECT.FIRMI and SVECT.FIRME;

END x_y_smoothing;
```

```
----------------------------------------------------------------------------

PROCESS z_tracker;

     < Level occupancy vertical tracker >

     Read report altitude data;
     Predict altitude:  <prediction to current scan>
     IF (mode C report hasn't been received)
          THEN Coast track;
          ELSE Calculate change in mode C report;
               SET valid altitiude data flag;
               Update z firmness;
               Adjust level occupancy bin size;
               IF (tracker is in startup period)
                    THEN PERFORM startup_smoothing;
                    ELSE IF (altitude transition has occurred)
                              THEN PERFORM altitude_transition_logic;
                              ELSE PERFORM nontransition_logic;

     Update track times in state vector;

END z_tracker;
```

```
--------------------------------------------------------------------------------

PROCESS z_tracker;

     < Level occupancy vertical tracker >

     ZR = surveillance data MODE C ALTITUDE;
       <prediction to current scan>
     SVECT.ZP = SVECT.ZS + (SYSVAR.CTIME - SVECT.TLUPD) * SVECT.ZDE;
     IF (surveillance data MODE C PRESENT EQ $FALSE)
          THEN SVECT.ZS = SVECT.ZP;
               SVECT.FIRMZ = MAX( 0, SVECT.FIRMZ - 1 )
          ELSE DZM = ZR - SVECT.ZPREV;
               SET SVECT.MCFLG;
               SVECT.FIRMZ = MIN( FIRMZ_MAX, SVECT.FIRMZ + 1 );
               DBINS = ABS( DZM ) / SYSTEM.Q;
               SVECT.SUCNT = SVECT.SUCNT + CNT_DELT;
               IF (SVECT.SUCNT LE XTONORM)
                    THEN PERFORM startup_smoothing;
                    ELSE IF (DZM NE 0.)
                              THEN PERFORM altitude_transition_logic;
                              ELSE PERFORM nontransition_logic;

     IF (SVECT.MCFLG EQ $TRUE)
          THEN SVECT.TMZ = SYSVAR.CTIME;
     SVECT.TLUPD = SYSVAR.CTIME;

END z_tracker;
```

5. INTERFACE MESSAGE PROCESSING

5.1 Message Classifications

The ATARS message interface is processed through six buffers:

- Non-surveillance
- Uplink
- ATC Coordination
- RAR
- Surveillance
- ATARS-to-ATARS

The contents of the messages exchanged through these buffers and the source responsible for message generation are discussed in this section.

5.1.1 Non-surveillance Messages

The essential fields for all the non-surveillance messages are shown in Tables 5-1 and 5-2. Certain messages are processed once per sector at the initiation of report processing. Other messages are processed by later tasks, as described below.

5.1.1.1 Outgoing Messages

The Data Link Capability Request Message is generated for a particular DABS aircraft. The response for this request is a Data Link Capability Reply Message. The ATARS Extended Data Request Message is generated for a particular DABS aircraft. The response for this request is an ATARS Extended Data Message. These messages provide ATARS with the aircraft's equipage, class of service and climb performance capability.

When an aircraft has entered the ATARS service area, a Start ATARS Service Message will be sent to the DABS sensor. The sensor will then begin to uplink the ATARS site ID each scan. The start message is generated in the Track Update Process. A Stop ATARS Service Message is sent to the sensor when the aircraft leaves the ATARS service area, or when the track is lost. This message is generated in geographical processing or in track update processing, for the case of a lost track.

When an aircraft which is BCAS equipped penetrates the ATARS service area beyond a designated ATARS-BCAS seam, the Squitter Lockout Message is sent to the DABS sensor. The sensor surveillance uplink will then inhibit squitters while the aircraft is inside this ATARS-only area. When an aircraft

TABLE 5-1

ATARS-TO-SENSOR MESSAGES

NON-SURVEILLANCE BUFFER

1. Data Link Capability Request (see Reference 8)

   Type Code

   DABS ID

2. ATARS Extended Data Request (see Reference 8)

   Type Code

   DABS ID

3. Start/Stop ATARS Service (see Reference 1)

   Type Code

   DABS ID

   ATARS Site ID (local or remote)

   Start/Stop Flag

   Note: Forward to remote sensor if indicated

4. Squitter Lockout (see Reference 1)

   Type Code

   DABS ID

   Start/Stop Flag

5. Set BCAS Sensitivity Level (see Reference 1)

   Type Code

   DABS ID

   SLC Field

6. ATARS Status (see Reference 1)

   Type Code

   Normal Operation/Failure Flag

TABLE 5-1
(Continued)

UPLINK MESSAGE BUFFER

1. Data Link Message Construction Output (Tactical Uplink)(see Reference 8)

    Header Field

        Type Code

        DABS ID

        Message Number

        Priority

        Expiration Time

        Sensor ID (local or remote)

    Message Field (MA) (see Reference 9, Para. 3.3.2.2)

        ADS Code

        Message

    Note: Remote sensor is expected to return the delivery notice to the sending ATARS site

TABLE 5-1
(Concluded)

ATC COORDINATION BUFFER (ATARS-to-ATC)

1. ATARS Operational Messages (see Reference 8)

   Type Code:

   a. Controller Alert

   b. ATERN Control Acknowledge

   c. FAZ Control Accept/Reject

   d. RAS Control Accept/Reject

   e. FAZ Data Base

   f. RAS Data Base

2. ATARS Status Messages (see Reference 8)

   Type Code:

   a. ATARS Green Condition

   b. ATARS Yellow Condition Codes

   c. ATARS Red Condition Codes

General Note:

ATARS does not generate the following messages as shown in Reference 1:

1. Uplink Message Cancellation Request (monolink)
2. Request for Downlink Data (monolink)
3. Track Data Request/Cancel Message
4. All "multilink" messages are deleted

TABLE 5-2

SENSOR-TO-ATARS MESSAGES

NON-SURVEILLANCE BUFFER

1. Data Link Capability Reply (see Reference 8)

    Type Code

    DABS ID

    Capabilities

        Equipage (ATARS and BCAS)

        Class of ATARS Service

2. ATARS Extended Data (see Reference 9, Para. 3.3.2.3.3)

    Type Code

    DABS ID

    Climb Performance Capability

3. Uplink Delivery Notice (see Reference 8)

    Type Code

    DABS ID

    Message Number

    Successful Delivery Flag

    Note: Forward to ATARS from remote sensor if required

4. Status (see Reference 1)

    Type Code

    Local Sensor Status

    Adjacent Sensor Status

    Adjacent ATARS Status

TABLE 5-2
(Continued)

<u>SURVEILLANCE BUFFER</u>

1. Correlated Surveillance Messages

Format Type:

a. DABS (see Table 3.3)

b. ATCRBS (see Table 3.4)

c. Radar (see Table 3.5)

<u>RAR BUFFER</u>

1. RAR Reply (Tactical Downlink)(see Reference 8)

Type Code

DABS ID

MB Field (see Reference 9, Para. 3.3.2.3.1)

<u>ATC COORDINATION BUFFER</u> (ATC-to-ATARS)

1. ATC Request (see Reference 8)

Type Code

Message Field (2 bits)

00	Neither radar-only nor non-mode C tracking desired
01	Not used
10	Non-mode C tracking desired and radar-only tracking not desired
11	Both non-mode C and radar-only tracking desired

TABLE 5-2
(Concluded)

2. ATARS Operational Messages (see Reference 8)

Type Code:

a. ATERN Control

b. FAZ Control

c. RAS Control

d. Request FAZ/RAS Data Base

3. ATARS Status Message (see Reference 8)

Type Code:

Request Full ATARS Status Control

General Note:

ATARS does not use the following messages as shown in Reference 1:

1. RAR Busy Message
2. Message Rejection/Delay Notice
3. Track Alert
4. ATCRBS ID Code

leaves this area, an end message is sent to the DABS sensor. These messages are implemented in geographical processing (Section 6.2.2).

When a BCAS aircraft enters the ATARS service area, ATARS will generate a BCAS Sensitivity Level Message to select desensitized BCAS logic thresholds. ATARS uses a site-specific area map to determine the applicable zone boundaries. This function is controlled by geographical processing. Its purpose is to allow ATARS to be the primary collision avoidance system in the ATARS service area.

An ATARS Status Message is generated each scan to state whether the site is operating normally or in a failure mode.

5.1.1.2 Incoming Messages

The Data Link Capability Reply Message and the ATARS Extended Data Message are generated by the sensor in reply to the Data Link Capability Request and ATARS Extended Data Request Messages. These messages provide the equipage, class of ATARS service, and the climb performance capability.

For each uplink message sent to the sensor, an Uplink Delivery Notice Message is returned to ATARS from the sensor. The details of this message are discussed in Section 5.1.2.

Once a scan a Status Message is sent to ATARS from the sensor. This includes information on both local and adjacent sensors and on adjacent ATARS site communication lines. This keeps ATARS current on all local and remote, sensor and site availability.

5.1.2 Uplink Messages

ATARS uplink messages for an individual DABS aircraft are delivered to the Uplink Message Buffer (Table 5-1) as an ordered set. The UPMES pointer in the aircraft State Vector contains the location in memory of this set of messages (UPLST). The position of each message within the set corresponds to its intended position in the desired uplink sequence (see also Section 16.2.1). The DABS sensor returns an uplink delivery notice for each message indicating its success or failure in delivery. All notices for an aircraft are delivered in one contiguous block, but not necessarily in the order of uplink. For this reason, delivery notices are numbered to correspond to the intended order of uplink. When the set of uplink delivery notices for each aircraft is processed, the message number field in each notice is matched with a message in the set identified by UPMES.

5.1.3 ATC Coordination Messages

An ATC facility and ATARS coordinate certain actions through an exchange of messages passed through the DABS sensor. These messages are discussed in Sections 5.1.3.1 and 5.1.3.2.

5.1.3.1 Messages from ATC

ATC may send three types of messages to ATARS as shown in the (ATC-to) SENSOR-to-ATARS Messages (Table 5-2). The ATC Request Message contains a message field which defines the processing status of non-mode C aircraft and radar-only reports within the ATARS algorithms. The exact coding of this message field is shown in Table 5-2. The remaining two types of messages are to request full ATARS status control and ATARS operational control.

5.1.3.2 Messages to ATC

ATARS may send two types of messages to ATC as shown in the ATARS-to-SENSOR (to-ATC) Messages (Table 5-1). The ATARS Operational Messages consist of controller alert, ATERN, FAZ or RAS information. The Controller Alert Message takes two forms: the Conflict Resolution Data Message, and the Resolution Notification Message. Both are discussed in Section 11. The ATARS Status Messages contain the red, yellow, or green codes as described in Section 18.2.2.

5.1.4 RAR Message

The RAR Reply (Tactical Downlink) Message contains all the RAR information from the RAR equipped aircraft. This message is read by the RAR Processing Task (Section 5.2). The message is included in the group of SENSOR-to-ATARS Messages (Table 5-2).

5.1.5 Surveillance Messages

The local and remote surveillance reports required by the ATARS Report Processing Task consist of three types: DABS reports, ATCRBS reports, and radar-only reports (Tables 3-3, 3-4, 3-5). These reports are included in the group of SENSOR-to-ATARS Messages (Table 5-2) and are discussed in Section 3.2.

5.1.6 ATARS-to-ATARS Messages

These messages (Table 5-3) provide direct communication between adjacent ATARS sites to establish coordination of conflict encounter resolution responsibilities. The ATARS multi-site messages should be processed on a priority basis by the communications function of the DABS sensor.

TABLE 5-3

ATARS-TO-ATARS MESSAGES

ATARS-to-ATARS BUFFER

1. Start/Stop Remote RAR Data (see Reference 1)

    Type Code

    Aircraft DABS ID

    Start/Stop Flag

    Neighboring ATARS Site ID

    ATARS ID of Site Requesting Data

    Only One Scan of RAR Data Requested Flag

2. Remote RAR Data Relay (see Reference 1)

    Type Code

    Aircraft DABS ID

    Remote Sensor ID

    RAR Data (MB field)

TABLE 5-3
(Continued)

3. Conflict Table Request, Claim, Handoff, Deletion (see Reference 1)

Type Code:

a. Conflict Table Request

b. Claim

c. Handoff

d. Deletion

Message Fields (for all Type Codes):

Sending Site ID

Destination Site ID

AC1 Type: DABS or ATCRBS

AC1 ID: (if Type = DABS), DABS Address
(if Type = ATCRBS), Surveillance File No., Mode 3/A Code and Position: RHO, THETA, Z in sending sensor's coordinates

AC2 Type: (Like AC1 Type)

AC2 ID: (Like AC1 ID)

TABLE 5-3
(Concluded)

4. Conflict Table Reply (see Reference 1)

Type Code

Requesting Site ID

Replying Site ID

AC1 Type: Same as in Message Fields above

AC1 ID: Same as in Message Fields above

AC2 Type: Same as in Message Fields above

AC2 ID: Same as in Message Fields above

Number of Conflict Tables (0, 1, or 2)

First Conflict Table (if any)

Second Conflict Table (if any)

The ATARS multi-site messages assigned the highest priority are the following:

Conflict Table Request
Conflict Table Claim
Conflict Table Handoff
Conflict Table Deletion
Conflict Table Reply.

The delivery of these messages from ATARS to the sensor communications processor for access to the communication lines must be expidited. A window of only 4/16 of a scan period is allowed for the two-way communication and processing of these multi-site messages (see Figure 3-1). This timing restriction is necessary to provide ATARS with timely multi-site data.

The remaining multi-site messages:

Start/Stop Remote RAR Data
Remote RAR Data Relay

are assigned a lower priority. The timing window for the exchange of these messages must not exceed 1/2 scan. The RAR reply is distributed to the RAR Processing Task from this buffer.

5.2 RAR Processing Task

The RAR Processing Task processes resolution advisory data read down each scan from each ATARS-equipped aircraft's Resolution Advisory Register (RAR). This information is used to coordinate resolution between ATARS sites and between ATARS and BCAS and to ensure compatibility of resolution advisories. This coordination supplements ground line coordination (Section 14.1), which may not be available, by relaying essential data for the creation and deletion of Pair Record information in Conflict Tables.

5.2.1 Remote Relay of RAR Data

In the event a site fails to read an aircraft's RAR data, it can request this data from a connected site. This is done by uplink delivery notice processing (Section 5.1) when own-site's resolution advisory is not delivered. The RAR Processing Task sends a reply with the requested data.

5.2.2 External Resolution Pair Record Updating

The RAR data enables creation, updating, and deletion of Pair Records for conflicts resolved by external sources, namely other ATARS sites and BCAS. The RAR data names the source of resolution and the resolution advisory, but does not identify the threat aircraft. Thus, in cases where ground line coordination does not also exist, the second aircraft in a pair will be marked as "unknown" identity. However, the essential information is present, namely the subject aircraft's resolution advisory, which acts as a constraint upon any additional advisories which own-site may select.

The external updating procedure operates in turn, upon all columns of the aircraft RAR data except own-site's column. Also, data from a connected site's column is not used if ground line coordination with that site has commenced.

When an aircraft's RAR is "empty," that is, no resolution advisories are present, an RAR data report is still prepared by the Report Processing Task (Section 4.4). External updating still runs, and deletes any existing Pair Records for the aircraft which show other sources in charge.

5.2.3 Internal Resolution Pair Record Updating

An ATARS site selects resolution advisories based on its knowledge of any prior advisories sent by other sources. Between the time of advisory selection and their delivery to the aircraft, another source may have delivered an unexpectedly incompatible advisory. Own-site's advisory, arriving later, would be rejected by the aircraft RAR. The Internal Updating Process must recognize this case and cause a recomputation of own-site's advisory. A related case occurs when own-site's advisory has been delivered first, and a BCAS wishes to resolve another conflict using an advisory incompatible with the ATARS advisory. Since active BCAS is limited to vertical resolution and would likely have no sensible alternative choice, ATARS must recognize this situation and change its advisory, thus allowing the BCAS advisory to be entered.

The internal updating procedure examines the uplinked RAR column that own-site sent to the aircraft. This column is the merger of all of own-site's pair resolutions for the aircraft. All "zeros" in the column are known to have been entered in the RAR. However, all "ones," representing intended advisories, must be tested against the prior contents of the rest of the RAR, to mimic the RAR rejection logic.

A composite column is formed from the logical OR of all other RAR columns. Then each bit set to one in the uplink column is tested for compatibility with the composite column. If any incompatibility is found, ATARS knows its advisory was rejected by the aircraft. Even after an ATARS advisory has been present, a BCAS negative advisory will be treated as a "prior constraint" by this process, and the ATARS recomputation logic will be performed.

For each of own-site's advisories accepted by the aircraft RAR, a Resolution Pair Acknowledgement List entry is generated. This list is processed by the controller alert Resolution Notification Task (Section 11).

5.3 Pseudocode for Interface Message Processing

The pseudocode follows for the two tasks described in this section. The Non-surveillance Message Processing Task calls the Remote Function Status Routine. This routine is contained in Section 17, Backup Mode. The RAR Processing Task calls the Remote RAR Start/Stop Data Routine, found in the previous task, and calls the Pair Record Deletion Routine, contained in Section 15.

AD-A104 147 MITRE CORP MCLEAN VA METREK DIV F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-1 FAA-RD-81-45-1 NL

3 OF 5
AD
A104147

PSEUDOCODE TABLE OF CONTENTS

PAGE

```
-------------------------------------------------------------------------------
TASK NON_SURVEILLANCE_MESSAGE_PROCESSING
   IN (non_surveillance, ATARS_to_ATARS, and ATC coordination buffer data,
         CREPD, RAPR parameters)
   OUT (state vector, system variables, messages to DABS)
   INOUT (conflict tables, remote list);

     <Process incoming messages in the buffers listed above>

     REPEAT WHILE (messages remain in input buffer);
          IF (an UPLINK DELIVERY NOTICE message)
               THEN PERFORM uplink_delivery_notice_processing;
          ELSEIF (a DATA LINK CAPABILITY REPLY OR ATARS EXTENDED DATA message)
               THEN PERFORM data_link_capability_processing;
          ELSEIF (a STATUS message)
               THEN CALL REMOTE FUNCTION STATUS;
          ELSEIF (an ATC REQUEST message)
               THEN PERFORM ATC_request_message_processing;
          ELSEIF (ATARS OPERATIONAL messages)
               THEN PERFORM ATC_to_ATARS_operational_message_processing;
          ELSEIF (an ATARS STATUS message)
               THEN PERFORM ATC_to_ATARS_status_message_processing;
          ELSEIF (a START/STOP REMOTE RAR DATA message)
               THEN CALL REMOTE_RAR_START/STOP_DATA with start/stop flag according to
                         message;
          ELSEIF (a REMOTE RAR DATA RELAY message)
               THEN Place REMOTE RAR DATA REPLY message in RAR buffer;
          OTHERWISE; <message not identified or hold for multi-site message
                         processing>
     ENDREPEAT;

END NON-SURVEILLANCE_MESSAGE_PROCESSING;




-------------- NON-SURVEILLANCE MESSAGE PROCESSING HIGH-LEVEL LOGIC ---------------
```

```
--------------------------------------------------------------------------------
TASK NON_SURVEILLANCE_MESSAGE_PROCESSING
   IN (non_surveillance, ATARS_to_ATARS, and ATC coordination buffer data,
        CREFD, RAERPARM)
   OUT (SVECT, SYSVAR, messages to DABS)
   INOUT (conflict tables, remote list);

     <Process incoming messages in the buffers listed above>
     REPEAT WHILE (messages remain in input buffer);
          IF (an UPLINK DELIVERY NOTICE message)
               THEN PERFORM uplink_delivery_notice_processing;
          ELSEIF (a DATA LINK CAPABILITY REPLY OR ATARS EXTENDED DATA message)
               THEN PERFORM data_link_capability_processing;
          ELSEIF (a STATUS message)
               THEN CALL REMOTE_FUNCTION_STATUS
                         IN (Status message)
                         INOUT (SYSVAR, SVECT, conflict tables);
          ELSEIF (an ATC REQUEST message)
               THEN PERFORM ATC_request_message_processing;
          ELSEIF (ATARS OPERATIONAL messages)
               THEN PERFORM ATC_to_ATARS_operational_message_processing;
          ELSEIF (an ATARS STATUS message)
               THEN PERFORM ATC_to_ATARS_status_message_processing;
          ELSEIF (a START/STOP REMOTE RAR DATA message)
               THEN CALL REMOTE_RAR_START/STOP_DATA
                         IN (AC ID, start/stop flag)
                         OUT (SVECT, message to DABS);
          ELSEIF (a REMOTE RAR DATA RELAY message)
               THEN Place REMOTE RAR DATA REPLY message in RAR buffer;
          OTHERWISE; <message not identified or hold for multi-site message
                         processing>
     ENDREPEAT;

END NON_SURVEILLANCE_MESSAGE_PROCESSING;


-------------- NON-SURVEILLANCE MESSAGE PROCESSING LOW-LEVEL LOGIC ---------------
```

```
-----------------------------------------------------------------------------------

PROCESS uplink_delivery_notice_processing;
<Delivery notice specifies aircraft, message number and if successfully delivered.>
    Match notice number to message in stored list for aircraft;
    REPEAT UNTIL (all subfields processed); <in MA field of message>
        IF (subfield type is Resolution)
            THEN IF (not success)
                   THEN Send message to remote ATARS requesting one-scan RAR data;
                        IF (Resolution Advisory newly selected last scan)
                          THEN remove Resolution Advisory from pair record;
                               Mark pair record for recomputation of Res. Adv. ;
        ELSEIF (subfield type is Prox or Threat)
            THEN IF (TA_class GT 0 AND subfield is part of Resolution message)
                      THEN;
                 ELSEIF (subfield is Prox part of Threat advisory message)
                      THEN;
                 ELSEIF (not an End message)
                      THEN IF (success)
                                THEN Save message type;
                                     SET End flag in PWILST entry;
                                ELSE;
                      ELSE;
        ELSEIF (subfield type is Terrain OR Airspace OR Obstacle)
            THEN SET End flag in PWILST entry;
                 IF (success)
                      THEN CLEAR First-time-transmitted bit in PWILST entry;
                      ELSE;
        ELSEIF (subfield type is Altitude Echo AND not success)
            THEN Set ALECT to uninitialized value;  <force ALEC next scan>
        ELSEIF (subfield type is Own Message AND not success)
            THEN Set OWNT to uninitialized value;  <force OWN next scan>
        OTHERWISE;<don't process start/end, start threat
                      or ATCRBS_TB subfields in message>
    ENDREPEAT;
END uplink_delivery_notice_processing;

-------------- NON-SURVEILLANCE MESSAGE PROCESSING HIGH-LEVEL LOGIC --------------
```

```
----------------------------------------------------------------------------
PROCESS uplink_delivery_notice_processing;
<process one notice. Message indicates aircraft, message number, and
                    successful delivery or not>
    Match message number in notice to message in same position in SVECT.UPMES list;
    REPEAT UNTIL (all subfields processed in message); <in UPMES list>
         IF (subfield type is Resolution)
              THEN IF (not success)
                        THEN Send RAR REQUEST message to site in SVECT.REMRAR,
                                  with OSCFLG set;
                             IF (SYSVAR.CTIME-PREC.TCMD LE SYSTEM.SCANT)
                                  THEN PREC.HMAN,VMAN= $NORES;
                                       IF (PREC.POSCMD EQ $DOUBLE)
                                            THEN PREC.POSCMD=$RCMDBL;
                                            ELSE PREC.POSCMD=$RCMSNG;
         ELSEIF (subfield type is Prox or Threat)
              THEN IF (TA_class GT 0 AND subfield is part of Resolution msg)
                        THEN;
                   ELSEIF (subfield is Prox AND msg type is 'Threat')
                        THEN;
                   ELSEIF (END=0)  <in PWILST entry>
                        THEN IF (success)
                                  THEN OLD_TYPE=TYPE; <in PWILST entry>
                                       SET END; <in PWILST entry>
         ELSEIF (subfield type is Terrain OR Airspace OR Obstacle)
              THEN SET END; <in PWILST entry>
                   IF (success)
                        THEN CLEAR PTAT; <in PWILST entry>
         ELSEIF (subfield type is ALEC AND not success)
              THEN SVECT.ALECT= uninitialized value;  <force ALEC next scan>
         ELSEIF (subfield type is OWN AND not success)
              THEN SVECT.OWNT= uninitialized value;  <force OWN next scan>
         OTHERWISE;<don't process start/end, start threat
                                       or ATCRBS_TB subfields>
    ENDREPEAT;
END uplink_delivery_notice_processing;
-------------- NON-SURVEILLANCE MESSAGE PROCESSING LOW-LEVEL LOGIC ---------------
```

```
-------------------------------------------------------------------------------

PROCESS data_link_capability_processing;

    IF (type code is DATA LINK CAPABILITY REPLY)
        THEN IF (DABS identity in CREFD)  <aircraft known to site>
            THEN Obtain state vector;
                Update BCAS and ATARS equipage;
                Update ATARS class of service;
        ELSEIF (DABS identity in CREFD) <aircraft known to site>
            THEN Obtain state vector;
                Update AC climb performance;
        ELSE;

END data_link_capability_processing;
```

```
PROCESS data_link_capability_processing;

     IF (message type code EQ DATA LINK CAPABILITY REPLY)
          THEN IF (DABS identity in CREFD)  <aircraft known to site>
                    THEN Update SVECT.ATSEQ;
                         Update SVECT.ACLASS;
     ELSEIF (DABS identity in CREFD) <aircraft known to site>
               <message is ATARS EXTENDED DATA>
               THEN Update SVECT.ACLP;

END data_link_capability_processing;
```

```
--------------------------------------------------------------------------------
PROCESS ATC_request_message_processing;

     < Non_mode C reports are ATCRBS reports without altitude data >
     < Radar only reports are surveillance reports from a 'skin' track only >

      < Update ATARS processing capabilities in response to ATC directive >

     IF (ATARS processing of non_mode C not desired)
          THEN CLEAR Non_mode C tracking and radar only flags;
          ELSE Process message field to indicate non_mode C tracking desired
                    OR both non_mode C and radar only tracking desired;

END ATC_request_message_processing;
```

```
-------------------------------------------------------------------------------------

PROCESS ATC_request_message_processing:

     < Non_mode C reports are ATCRBS reports without altitude data >
     < Radar only reports are surveillance reports from a 'skin' track only >

      < Update ATARS processing capabilities in response to ATC directive >

     IF (ATARS processing of non_mode C not desired)
          THEN CLEAR SYSVAR.ATCNMC and SYSVAR.ATCROR;
          ELSE Copy message field (first bit) into SYSVAR.ATCNMC and
               (second bit) into SYSVAR.ATCROR;

END ATC_request_message_processing;
```

```
----------------------------------------------------------------------------------------

PROCESS ATC_to_ATARS_operational_message_processing;

      < Process ATC-to-ATARS operational message >

     IF (ATERN CONTROL message)
          THEN Implement the advised minimum altitude below which ATARS
                    will not issue descend resolution advisories;
               Send ATERN CONTROL ACKNOWLEDGEMENT message to ATC;
     IF (FAZ CONTROL message)
          THEN Implement new FAZ maps in system;
               Send FAZ CONTROL ACCEPT/REJECT message to ATC;
     IF (RAS CONTROL message)
          THEN Implement new RAS maps in system;
               Send RAS CONTROL ACCEPT/REJECT message to ATC;
     IF (REQUEST FAZ/RAS DATA BASE message)
          THEN Send reply to ATC regarding FAZ/RAS DATA BASE;

END ATC_to_ATARS_operational_message_processing;
```

```
--------------------------------------------------------------------------------------

PROCESS ATC_to_ATARS_operational_message_processing;

     < Process ATC-to-ATARS operational message >

     IF (ATERN CONTROL message)
          THEN RAEBPARM.ATERN EQ value indicated in message;
               Send ATERN CONTROL ACKNOWLEDGEMENT message to ATC;
     IF (FAZ CONTROL message)
          THEN Implement new FAZ maps in system;
               Send FAZ CONTROL ACCEPT/REJECT message to ATC;
     IF (RAS CONTROL message)
          THEN Implement new RAS maps in system;
               Send RAS CONTROL ACCEPT/REJECT message to ATC;
     IF (REQUEST FAZ/RAS DATA BASE message)
          THEN Send reply to ATC containing FAZ DATA BASE and/or RAS DATA BASE;

END ATC_to_ATARS_operational_message_processing;
```

```
--------------------------------------------------------------------------------

PROCESS ATC_to_ATARS_status_message_processing;

      < Process the ATC request for full ATARS status control >

     Indicate to the Performance Monitor Task the ATARS STATUS messages
     containing the ATARS condition codes are to be sent to ATC;

END ATC_to_ATARS_status_message_processing;
```

```
---------------------------------------------------------------------------------------

PROCESS ATC_to_ATARS_status_message_processing;

     < Process the ATC request for full ATARS status control >

    Update SYSVAR.STATMSG according to request;

END ATC_to_ATARS_status_message_processing;
```

```
--------------------------------------------------------------------------------------

ROUTINE REMOTE_RAR_START/STOP_DATA
   IN  (AC ID, start/stop flag)
   OUT (State vector, message to DABS);

     IF (stop flag set)
          THEN Clear remote RAR relay register;
               IF (not giving ATARS service to aircraft)
                    THEN IF (aircraft on remote list AND not in a conflict table)
                              THEN Delete remote list entry for this aircraft;
                         Send STOP ATARS SERVICE message to DABS;

          ELSE IF (not giving ATARS service to named aircraft)
                    THEN IF (no state vector or remote entry exists for aircraft)
                              THEN Enter aircraft on remote list;
                         Send START ATARS SERVICE msg to DABS using ID of requesting site;
               Save requesting site_ID in RFTRAR;
               Update one-scan flag according to request;
               Send TACTICAL UPLINK messages to DABS; <res. advs contain
                    ID of originating site>

END REMOTE_RAR_START/STOP_DATA;
```

```
--------------------------------------------------------------------------------------

ROUTINE REMOTE_RAR_START/STOP_DATA
   IN (AC ID, start/stop flag)
   OUT (SVECT, message to DABS);

     IF (stop flag set in START/STOP REMOTE RAR DATA message)
          THEN CLEAR SVECT.RETRAR;
               IF (not giving ATARS service to aircraft)
                    THEN IF (aircraft on remote list AND not in a conflict table)
                              THEN Delete remote list entry for this aircraft;
                         Send STOP ATARS SERVICE message to DABS;

          ELSE IF (not giving ATARS service to named aircraft)
                    THEN IF (no state vector or remote entry exists for aircraft)
                              THEN Enter aircraft on remote list;
                         Send START ATARS SERVICE msg to DABS using ID of requesting site;
               Save requesting site_ID in SVECT.RETRAR;
               Update  SVECT.OSCFLG;
               Send TACTICAL UPLINK messages to DABS; <res. advs contain
                    ID of originating site>

END REMOTE_RAR_START/STOP_DATA;
```

```
--------------------------------------------------------------------------------

STRUCTURE RAPVBL
  GROUP misc

    BIT COMPATIBLE                              <advisory compatible with RAR contents>

ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------

TASK RAR_PROCESSING

   IN (RAR Reports, state vectors)
   OUT (remote messages, resolution pair acknowledgment list)
   INOUT (conflict tables);

     REPEAT WHILE (any RAR reports in RAR buffer); <including "empty RAR" reports>
          PERFORM RAR_remote_relay;
          IF (aircraft in ATARS service or being dropped)
               THEN PERFORM external_updating;
                    IF (resolution message sent last scan)
                         THEN PERFORM internal_updating;
                    IF (aircraft in a conflict table)
                         THEN Determine displayed advisories by merging pair records;
     ENDREPEAT;

END RAR_PROCESSING;
```

---------------------- RAR PROCESSING TASK HIGH-LEVEL LOGIC ----------------------

```
---------------------------------------------------------------------------------

TASK RAR_PROCESSING

   IN (RAR Reports, state vectors)
   OUT (remote messages, Resolution Pair Acknowledgment List)
   INOUT (conflict tables);

     REPEAT WHILE (any RAR reports in RAR buffer);
          PERFORM RAR_remote_relay;
          IF (SVECT.ATSS EQ $TRUE OR SVECT.DRATS EQ $TRUE)
               THEN PERFORM external_updating;
                    IF (Resolution Message is in SVECT.UPMES list)
                         THEN PERFORM internal_updating;
                    IF (SVECT.CTPTR NE $NULL)
                         THEN Find displayed advisories by merging pair records
                               for subject AC;
                              Save displayed advisories in CTENTRY.HMAND, VMAND:
     ENDREPEAT;

END RAR_PROCESSING;

---------------------- RAR PROCESSING TASK LOW-LEVEL LOGIC ----------------------
```

```
--------------------------------------------------------------------------------
PROCESS RAR_remote_relay;

     IF (another site is to get RAR data)
          THEN relay RAR data to site specified;
          IF (one-scan flag set)
               THEN CALL REMOTE_RAR_START/STOP_DATA specifying "stop";

END RAR_remote_relay;
```

```
-------------------------------------------------------------------------------------

PROCESS RAR_remote_relay;

    IF (SVECT.RETRAR NE 0)
        THEN relay RAR data to site specified in SVECT.RETRAR;
        IF (SVECT.OSCFL EQ $TRUE)
            THEN CALL REMOTE_RAR_START/STOP_DATA
                    IN (AC ID, flag="stop")
                    OUT (SVECT, message to DABS);

END RAR_remote_relay;
```

```
--------------------------------------------------------------------------------
PROCESS external_updating;

     <Decode RAR report. Interpret as six columns of aircraft's RAR.
      Update own data base.>

     REPEAT UNTIL (all six RAR columns processed);
          IF (column is for other ATARS or BCAS AND column is empty)
               THEN REPEAT WHILE (any pair records showing that site in charge);
                         CALL PAIR_RECORD_DELETION;
                    ENDREPEAT;

          ELSEIF (column is for BCAS)    <own or other columns>
               THEN Create/update pair rec. showing intruder 'unknown',
                     BCAS in charge;
                    PERFORM resolution_pair_acknowledgment_entry_generation;

          ELSEIF (column is other ATARS site)
               THEN Look for pair rec(s) for this AC with same site in charge;
                    IF (no such pair record found)
                         THEN Create pair record showing intruder 'unknown';
                    ELSEIF (site not connected)
                         THEN Replace old pair rec(s) with current resolution(s);
                    ELSEIF (any such pair records found with both AC known)
                         THEN;  <don't use RAR data, ground lines OK>
                         ELSE Replace old pair rec(s) with current resolution(s);
                            <not using ground lines yet, use RAR data>

          OTHERWISE;   <don't act here on own site's column>
     ENDREPEAT;

END external_updating;
```

```
--------------------------------------------------------------------------------
PROCESS external_updating:

    REPEAT UNTIL (all six columns processed);
        IF (column is for other ATARS or BCAS AND column is empty)
            THEN REPEAT WHILE (any pair recs with PREC.ATSID=site ind. by column):
                    CALL PAIR_RECORD_DELETION
                      IN (ACID1, ACID2, PRPTR)
                      INOUT (Conflict tables, State Vectors);
                ENDREPEAT;

        ELSEIF (fifth or sixth column)  <own or other BCAS>
            THEN Create/update pair record so that PREC.ac2.PAC=$UNK
                                                  and PREC.ATSID=*BCAS;
                PERFORM resolution_pair_acknowledgment_entry_generation;

        ELSEIF (column is other ATARS site)
            THEN Look for pair rec(s) for this AC with PREC.ATSID=same site;
                IF (no such pair record found)
                    THEN Create pair record with PREC.AC2.PAC=$UNK;
                ELSEIF (site not connected)
                    THEN PREC.PHMAN,PVMAN =current resolution(s);
                ELSEIF (any such pair records found with both PREC.PAC NE $UNK)
                    THEN;  <don't use RAR data, ground lines OK>
                    ELSE PREC.PHMAN,PVMAN = current resolution(s);
                      <not using ground lines yet, use RAR data>

        OTHERWISE;   <don't act here on own site's column>
    ENDREPEAT;

END external_updating;
```

```
PROCESS internal_updating;

     <Test bits of uplinked column to see what got into RAR.>

     Set test column to all zeroes;
     Form logical OR of other ATARS sites' columns and BCAS columns;
     Extract uplink column from Resolution Message in UPMES;

     REPEAT WHILE (any more bits set in uplink column);
          PERFORM compatibility;
          IF (compatible)
               THEN SET same bit in test column;
               ELSE;
     ENDREPEAT;  <now test column represents AC RAR column>

     REPEAT WHILE (any more pair records for subj. AC with own site in charge):
          IF (null RA shown for subject AC)
               THEN CLEAR send flag for subject AC;
                    IF (other AC send flag not set)
                         THEN CALL PAIR_RECORD_DELETION;
          ELSEIF (bit not set in test col. matching RA for subj AC)  <either part of
                                                                       compound RA>
               THEN IF (RA newly selected last scan)
                         THEN Remove that RA from pair record;
                              <remove only part whose bit not set>
                    Signal pair needs recomputation;
          OTHERWISE    <RA accepted by subject RAR>
                    PERFORM resolution_pair_acknowledgment_entry_generation;
     ENDREPEAT;

END internal_updating;
```

```
------------------------------------------------------------------------

PROCESS internal_updating;

     <Test bits of uplinked column to see what got into RAR.>

    Set test column to all zeroes;
    Form logical OR of other ATARS sites' columns and BCAS columns;
    Extract uplink column from Resolution Message in SVECT.UPMES list;
    REPEAT WHILE (any more bits set in uplink column);
        PERFORM compatibility;
        IF (COMPATIBLE set)
            THEN SET same bit in test column;
            ELSE;
    ENDREPEAT;  <now test column represents AC RAR column>
    REPEAT WHILE (any more pair records with PREC.ac1.PAC or ac2.PAC EQ subj. AC
                                AND PREC.ATSID=$OWNID)
        IF (PREC.PHMAN or PREC.PVMAN = $NULLRES for subj. AC)
            THEN CLEAR PREC.SEND for subject AC;
                IF (other AC PREC.SEND=$FALSE)
                    THEN CALL PAIR_RECORD_DELETION
                            IN (ACID1, ACID2, PRPTR)
                            INOUT (Conflict Tables, State Vectors);
        ELSEIF (bit not set in test col. matching RA for subj AC)  <either part of
                                                                    compound RA>
            THEN IF (RA newly selected last scan)
                    THEN Remove that RA from pair record;
                            <Remove only part whose bit not set>
                IF (PREC.POSCMD EQ $DOUBLE)
                    THEN PREC.POSCMD=$RCMDBL;  <recomputation, double advs>
                    ELSE PREC.POSCMD=$RCMSNG;  <recomputation, single advs>
        OTHERWISE    <RA accepted by subject RAR>
                PERFORM resolution_pair_acknowledgment_entry_generation;
    ENDREPEAT;

END internal_updating;
```

```
---------------------------------------------------------------------------------

PROCESS compatibility;

     IF (bit in uplink column is positive horizontal)
          THEN IF (no opposite-sense horizontal bits set in RAR composite column)
                    THEN indicate compatible;
                    ELSE indicate not compatible;

     ELSEIF (bit in uplink column is negative horizontal)
          THEN IF (opposite-sense positive horizontal bit
                     not set in RAR composite column)
                    THEN indicate compatible;
                    ELSE indicate not compatible;

     ELSEIF (bit in uplink column is positive vertical)
          THEN IF (no opposite-sense vertical bits set in RAR composite column)
                    THEN indicate compatible;
                    ELSE indicate not compatible;

     OTHERWISE < bit in uplink column is negative vertical or VSL. >
               IF (opposite-sense positive vertical bit
                     not set in RAR composite column)
                    THEN indicate compatible;
                    ELSE indicate not compatible;

END compatibility;
```

```
PROCESS compatibility;

    IF (bit in uplink column is positive horizontal RA)
         THEN IF (no opposite-sense horizontal bits set in RAR composite column)
                   THEN SET COMPATIBLE;
                   ELSE CLEAR COMPATIBLE;

    ELSEIF (bit in uplink column is negative horizontal RA)
         THEN IF (opposite-sense positive horizontal bit not set in compos. column)
                   THEN SET COMPATIBLE;
                   ELSE CLEAR COMPATIBLE;

    ELSEIF (bit in uplink column is positive vertical RA)
         THEN IF (no opposite-sense vertical bits set in RAR composite column)
                   THEN SET COMPATIBLE;
                   ELSE CLEAR COMPATIBLE;

    OTHERWISE < bit in uplink column is negative vertical or VSL RA>
              IF (opposite-sense positive vertical bit not set in compos. column)
                   THEN SET COMPATIBLE;
                   ELSE CLEAR COMPATIBLE;

END compatibility;
```

```
--------------------------------------------------------------------------------

PROCESS resolution_pair_acknowledgment_entry_generation;


        <Generate RPALST entry from pair record.>


    Link new RPALST entry;  <don't need to check for duplication>

    Copy AC ID's and RA's from pair record;

    Enter current time;

    Indicate whether pair resolved by BCAS or ATARS;

END resolution_pair_acknowledgment_entry_generation;
```

```
--------------------------------------------------------------------------------

PROCESS resolution_pair_acknowledgment_entry_generation;

     Link new RPALST entry;
     RPALST.ac1.ID=PREC.ac1.PAC;
     RPALST.ac2.ID=PREC.ac2.PAC;
     RPALST.ac1.RES=PREC.ac1.PHMAN,PVMAN;
     RPALST.ac2.RES=PREC.ac2.PHMAN,PVMAN;
     RPALST.TIME=SYSVAR.CTIME;
     IF (PREC.ATSID EQ $BCAS)
          THEN CLEAR RPALST.SOURCE;      <BCAS  resolution>
          ELSE SET RPALST.SOURCE;        <ATARS resolution>

END resolution_pair_acknowledgment_entry_generation;
```

6. AIRCRAFT PROCESSING

6.1 New Aircraft Processing

The selection of new aircraft to be added to the ATARS data base is made in the Report Processing and Track Processing Tasks. These choices are conveyed to the New Aircraft Processing Task through the XINIT List. The purpose of this task is to add all aircraft on this list to the X-list, EX-list, and ATARS Sector List. The task will also initialize all parameters in the State Vector that are used in subsequent ATARS processing tasks.

The XINIT List is a linked list of all aircraft that have been designated for addition to the X-list or the EX-list, and to the sector list by the Track Processing Task for a particular ATARS sector. The XINIT List has a pointer to the head of the list and is linked, in one direction only, through the NEXTX position in the state vectors. The use of NEXTX is legitimate at this time because this field is not used for those aircraft not yet added to the X-list or the EX-list. New Aircraft Processing sets the NEXTX and PREVX pointers for each aircraft in the XINIT List to include the State Vectors in the forward and backward linked X-list or EX-list. The NEXTA pointer is set to include the next aircraft State Vector in the ATARS Sector List. These pointers are set using the X/EX-list Initial Entry Routine which is described in Section 6.1.3.

6.1.1 X-list and EX-list of Aircraft

The X-list and the EX-list are two lists of aircraft State Vectors which are ordered on the x coordinates of the aircraft with the DABS sensor at the origin. The X-list includes all aircraft whose altitude is below a threshold altitude, and whose speed is below an upper velocity limit. All other aircraft are on the EX-list. The position of the aircraft on these two lists is checked when the particular ATARS sector is being processed and it is updated if required.

Initial placement of aircraft into either list is described in Initial Entry of Aircraft Into the X-list, EX-list and Sector List (Section 6.1.3). The process of maintaining their current position on the lists is described in Sector List, X-list, and EX-list Updating (Section 6.2.3).

6.1.2 Sector List of Aircraft

The ATARS Sector List is a list of all aircraft State Vectors in an ATARS sector. There is one sector list for each ATARS sector.

The executive control keeps a table containing the start point for each sector's list. The sector list is used to expedite the selection of aircraft to be processed in each ATARS sector. All the tasks which need to look at every aircraft in a sector access these sector lists.

6.1.3 Initial Entry of Aircraft Into the X-list, EX-List, and Sector List

Both the X-list and the EX-list are modified to permit expeditious entry of new aircraft into these lists. This modification takes the form of seeding the ordered lists with dummy State Vectors which have fixed x coordinates. A cross-reference permits a direct means of locating the dummy State Vector nearest to a given x coordinate. The cross-reference takes the form of an array of pointers linking the known x coordinates. The dummy State Vectors are called signposts and contain, as a minimum, fields for the NEXTX and PREVX pointers, a field for the x value of the signpost, and flag to identify signposts, SPIDFG. SPIDFG will be permanently set for all signposts and reset for all aircraft State Vectors. This flag is used to expedite signpost identification in the Coarse Screen Processing Task.

To enter new aircraft into either list, a determination is first made if the aircraft is in the hub area. If so, the hub flag (HUBFLG) is set. If the aircraft qualifies for the EX-list, the aircraft is entered on the EX-list, and the EXFLG flag is set; otherwise, it is entered on the X-list and the EXFLG is reset. Both of these lists initially consist only of the signposts linked together. The NEXTX and PREVX pointers of the two terminal signposts in both lists are set to null.

The new aircraft are also linked to the particular ATARS sector list in the X-list or the EX-list corresponding to the sector identification.

The ATARS sector thread is linked only in the forward direction using a pointer NEXTA. If the new aircraft becomes the first aircraft in a sector thread, the executive program must be notified so that its SIDSPX and SIDSPE table is updated to reflect this change. When threading new aircraft into a sector list, care must be taken to keep the previous aircraft position on the list, as a backwards pointer is not required for any other ATARS Sector List processing and thus is not part of the State Vector.

6.2 Aircraft Update Processing

The function of the Aircraft Update Processing Task is to update the position and velocity coordinates, determine the ATARS service zone through geographical processing (Section 6.2.2) and update the position of those aircraft on the X-list, EX-list, and sector list (Section 6.2.3) in the sector designated by the executive control. The position coordinates of the hub area aircraft are updated as part of this task.

Aircraft update processing operates on a particular sector list of aircraft threaded in the X-list and EX-list. The start pointers (SIDSPX, SIDSPE) for the sector lists are in a table maintained by the executive control and are updated in the list update processing (Section 6.2.3). The table holds the start pointers for every sector which contains aircraft on the X-list or the EX-list and null pointers for sectors which contain no aircraft. The start pointer identifies the State Vector of the aircraft that starts the string of aircraft for a sector.

The position coordinate update is achieved by a linear projection from the aircraft's predicted position, for a time equal to the difference between the sector time (TEN) and the aircraft time of prediction (TMP) using the aircraft external velocity components. All aircraft are then presented for sector processing on a common basis. It is important to note that the only coordinates used in ATARS sector processing after the Aircraft Update Processing Task is complete are the position and velocity coordinates set in this task. The updated coordinates are used to update the geographical zone if the ATARS service flag is set. The updated coordinates are also used to update the position in the X-list, EX-list, and sector list for the subject aircraft before continuing with the next aircraft in the X-list or the EX-list. This process continues until all aircraft in the sector thread for both the lists are completed.

The variables TEN and TMP will have a fixed number of bits. These variables will recycle to zero when the time overflows the number of available bits. Hence in subtracting time variables, here and throughout the ATARS processing both variables should be expressed with common higher order bits.

Special attention must be given to the structuring of the update processing task. Although the X-list and EX-list are used to access the aircraft for updating, one of the steps in updating is to reposition the aircraft in the lists. The next aircraft in the list is saved before the current aircraft is repositioned in either list. However, even with this procedure, an aircraft which has moved up the list by skipping one or more aircraft would be accessed a second time for updating in the same sector list. To prevent this duplicate updating, the XUPFL flag in the State Vector is set the first time the aircraft is accessed for updating and is read to prevent duplicate updating in the second access. The XUPFL flag is cleared once per sector when the aircraft is accessed in the Coarse Screen Processing Task.

The hub area position coordinate update is exercised at the start of every quarter scan (i.e., when the antenna is at 0^o, 90^o, 180^o, 270^o). The hub area (Figure 3-2) is defined as a circle of given radius centered at the DABS sensor. First, the signpost (explained in Section 6.1.3) on the X-list or the EX-list nearest to the negative value of the hub area radius is located. The lists are searched forward until the first signpost beyond the positive value of the hub radius is reached. All of the aircraft identified as being in the hub on either list between these negative and positive signposts are afforded hub processing. The position and velocity coordinates of these aircraft are updated using the same technique described above for the regular sector aircraft position and velocity updating. This processing maintains a more current position on both lists for the aircraft located near the DABS sensor where ATARS sectors are not very wide. This is necessary because aircraft may change sectors rapidly in the hub area and this affords a more timely identification of a possible conflict in the ATARS processor.

Again, the XUPFL flag is used to prevent duplicate updating. The flag is reset in this task instead of waiting for the Coarse Screen Processing Task because the aircraft identified in the hub area may be from any sector, not just the particular sector being processed at this time. Therefore, only those XUPFL flags associated with hub processing would be reset at this pass through the ATARS processor.

6.2.1 ATARS Service Map

Every ATARS site stores a map indicating the boundaries of the area for which ATARS service is provided. The location of the boundaries differ depending whether or not another ATARS site is near enough to cover an adjacent area. The two cases are called the "single-site" coverage and the "multi-site" coverage cases. However, a site may use both kinds of coverage at once, as in the example depicted in Figure 6-1. This hypothetical site has no neighbor to its West, and so uses single-site boundaries on that side. Another site is located nearby to the East, and so multi-site coverage is assigned to that side.

The single-site boundaries give ATARS as much coverage as possible. The surveillance mask, ATARS service map, and Domino coverage all share a common boundary. This boundary may be identical with the extent of DABS surveillance coverage (and thus need not be an explicit boundary, although some values must be assigned to the map); or it may be chosen to delimit an area within DABS coverage, if DABS position data is not believed to be sufficiently accurate at long range.

The multi-site boundaries serve to partition the ATARS processing load among nearby sites. Overlap is provided to assure continuous coverage during boundary crossings, smooth handoff of responsibility between sites, and to allow for small differences in the sites' perceptions of the boundary.

Figure 6-1 shows several boundaries for the subject site. The Domino/Surveillance boundary serves the following functions:

1. Report Processing (Section 4.4) makes a simple rho-theta check used to accept or reject surveillance reports.

2. ATARS tracks are maintained on aircraft within this area.

3. Although ATARS service is not provided to aircraft outside the ATARS service area (see below), the Domino Routine (Section 13.4.2.1) considers aircraft within the Domino boundary when selecting resolution advisories for aircraft receiving ATARS service.

The Domino/Surveillance boundary is stored as one or more azimuth sectors with a range boundary corresponding to each sector. These sectors are distinct from DABS sensor or ATARS processing sectors. This boundary must be wide enough to provide the desired Domino coverage for conflicts near the limits of the ATARS service area. It is also desirable to include surveillance coverage for the site's Backup Service Area (see Section 17.3.1). If such coverage is provided, any backup mode switchover would be smooth, since tracks would be available.

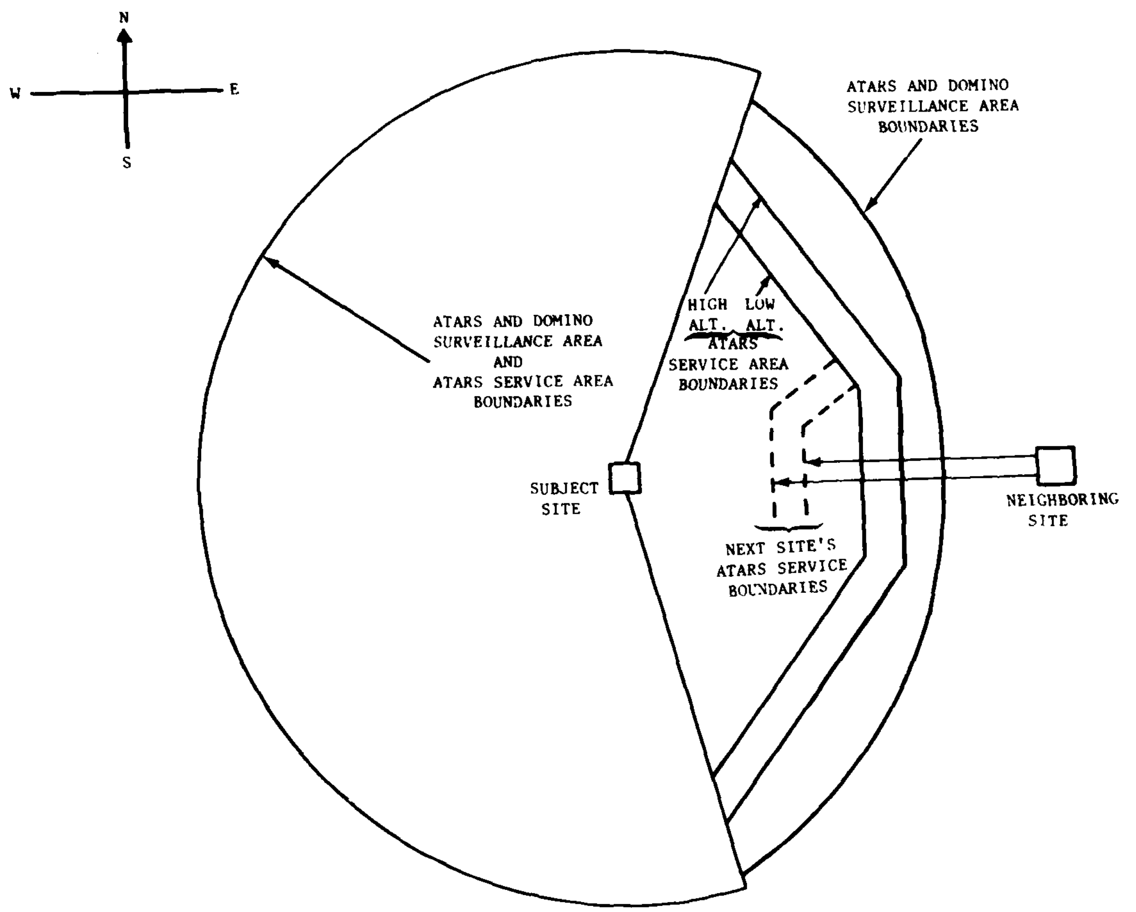

N
W
E
S
ATARS AND DOMINO SURVEILLANCE AREA BOUNDARIES
ATARS AND DOMINO SURVEILLANCE AREA AND ATARS SERVICE AREA BOUNDARIES
HIGH ALT.
LOW ALT.
ATARS SERVICE AREA BOUNDARIES
SUBJECT SITE
NEIGHBORING SITE
NEXT SITE'S ATARS SERVICE BOUNDARIES

FIGURE 6-1
ATARS SERVICE MAPS FOR SINGLE-SITE AND MULTI-SITE COVERAGE

The ATARS service area contains aircraft for which traffic advisory and resolution advisory services are provided. A discussion of Normal and Backup areas is provided in Section 17.3.1. The following description applies to whichever map is in use at a given time.

The service mask is implemented as a pair of convex polygons, one for aircraft from the lower limit of DABS coverage up to and including altitude HZONE, and the other for aircraft above altitude HZONE. These polygons may have as many vertices as are needed. For example, to implement multiple site coverage in a large area, intricate shapes may be required to provide the necessary overlap (seams) and divide the expected traffic and processing load.

It is essential that wherever two ATARS sites have a common boundary, their normal mode ATARS service areas overlap by no less than a specified distance. This overlap is called a "seam" and is given a width sufficient to allow nominal warning time for any pair of aircraft on opposite sides of the boundary. The seam is implemented by extending each site's area a distance DSEAMH (or DSEAML) past the nominal boundary of the high (or low) altitude map. Then the sites' coverages overlap by 2*DSEAMH for the high altitude masks. A seam is illustrated in Figure 6-2.

The following algorithm is used to determine whether the tracked aircraft position (XT, YT) lies within a convex polygon. This figure will be described by the NVERT vertices, (X_1,Y_1), $(X_2,Y_2),\ldots,(X_{NVERT},Y_{NVERT})$. The vertices are numbered in a counterclockwise manner starting at any vertex as shown for the example 4-sided area of Figure 6-3. Form all vectors joining adjacent vertices in a counterclockwise direction as indicated by the vectors V_1 through V_4. Also form all vectors joining the vertices to the point (XT, YT). Let these vectors be designated VT_1 through VT_4. Then the point (XT, YT) is internal to the zone if and only if all vector cross products $VT_1{*}V_1, VT_2{*}V_2,\ldots,VT_{NVERT}{*}V_{NVERT}$, are negative. By precomputing several constants which are functions of the vertex coordinates, the number of calculations required to perform the test is reduced.

A typical vector cross product, $VT_i{*}V_i$ is,

$$VT_i = (XT-X_i)*(Y_{i+1}-Y_i)-(YT-Y_i)*(X_{i+1}-X_i).$$

This can be written,

$$VT_i{*}V_i = XT{*}DY_i - YT{*}DX_i+K_i$$

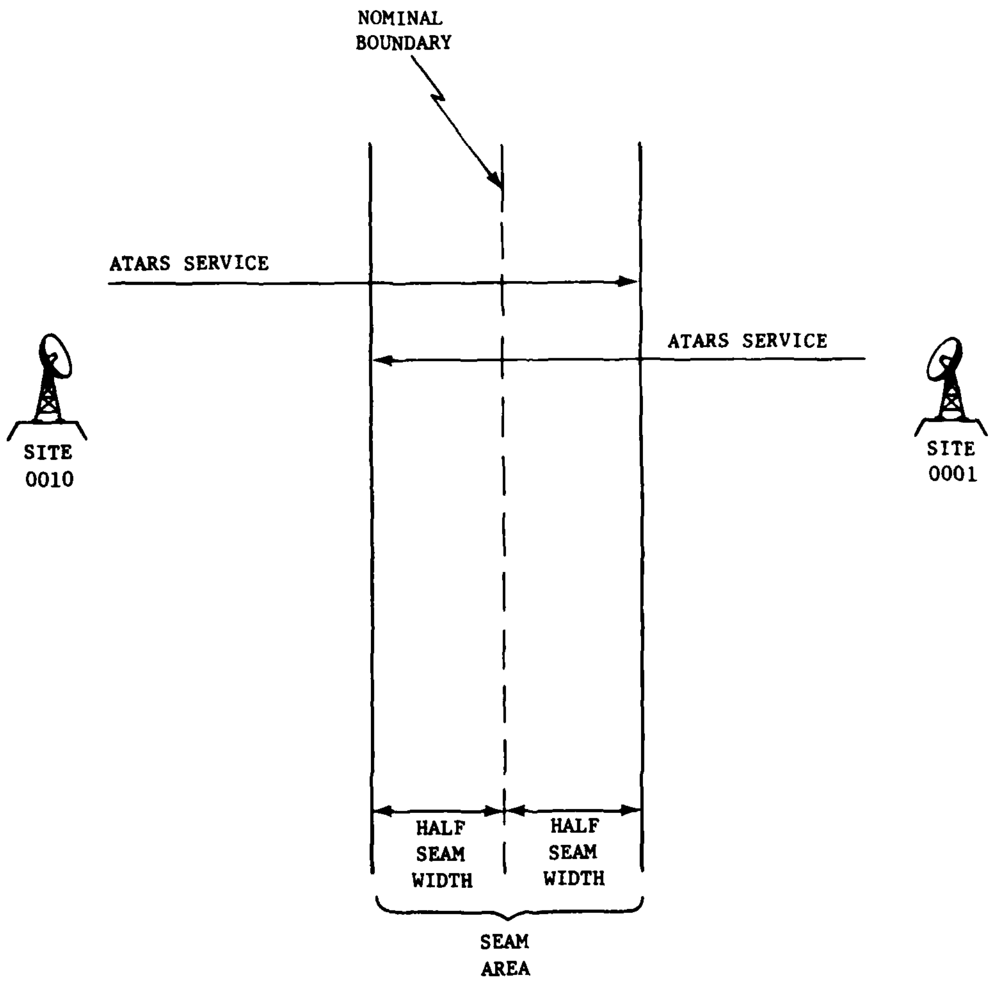


FIGURE 6-2
ATARS BOUNDARIES AND SEAM AREA

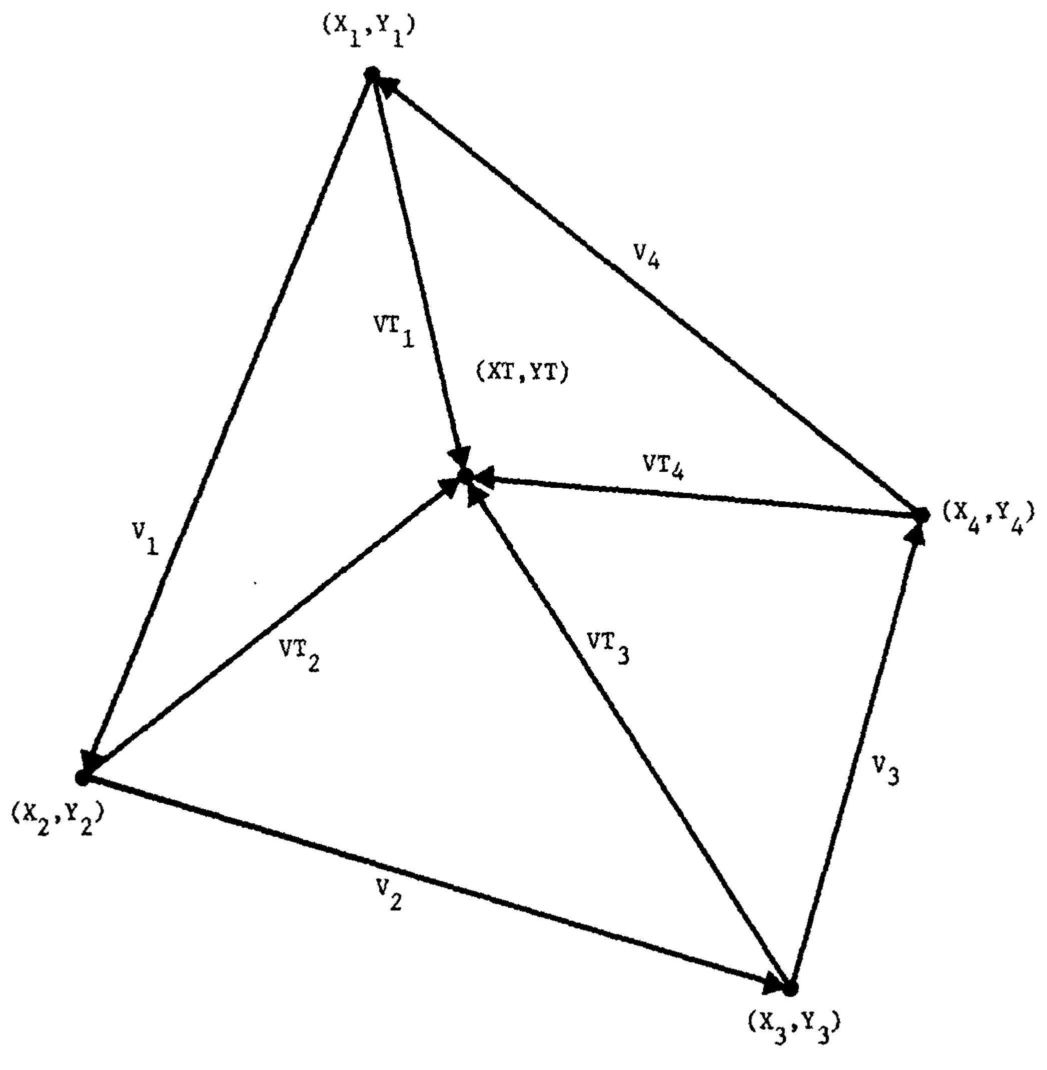


FIGURE 6-3
DETERMINING WHETHER THE POINT (XT, YT)
LIES WITHIN A POLYGON

where, $DX_i = X_{i+1} - X_i$
$DY_i = Y_{i+1} - Y_i$
$K_i = Y_i*DX_i - X_i*DY_i$

All the DX, DY and K constants for each map are precomputed and stored in memory. Then the expression,

$$F(i) = XT*DY_i-YT*DX_i+K_i$$

is evaluated repeatedly from i= 1 to NVERT. If any term F(i) is positive, the aircraft is outside the polygon. If all are negative, it is inside.

6.2.2 Geographical Processing

This process tests an aircraft's position to determine whether it is within the site's ATARS service area. In the special case when the site is in the Backup-Master mode, (see Section 17.3.2) separate tests are performed for the site's Center Zone and its Backup service area. Note that the Report Processing Task (Section 4.4) has already performed a quick test on the position and has set ATSS in the State Vector if the aircraft is within the rho-theta mask (larger than the service area). If geographical processing now finds the aircraft outside the service area, it clears the service flag.

The State Vector GEOG contains all site ID bits read down (for DABS aircraft) in the latest surveillance report. If the aircraft is entering this site's ATARS service area, own site ID bit will not have been sent up yet. The bit is set here so that the Seam Pair Task will treat the aircraft properly.

For aircraft within the service area, additional processing is performed according to their equipage. All BCAS-equipped aircraft are sent Sensitivity Level control (SLC) messages determined by a stored map. The message is sent only when SLC should change. All DABS-equipped aircraft are sent a Squitter Lockout Message when within a designated area. This message is sent only when the lockout status changes. The DABS sensor sends SLC and lockout control data each scan based on the last messages sent from ATARS.

When an aircraft leaves the ATARS service area, the map test detects the condition and clears the service flag. The site's ID bit is removed from GEOG. This prevents Seam Pair Task from passing the pair to Master Resolution Task. If the aircraft is seen by any connected sites (as indicated in GEOG) then handoff messages are sent to those sites to enable a transfer of responsibility.

6.2.3 Sector List, X-list, and EX-list Updating

The position of all the aircraft in a sector are updated on the sector list, X-list, and EX-list. If necessary, an aircraft may be moved from the X-list to the EX-list or from the EX-list to the X-list. Aircraft which were on the EX-list on the last scan have been marked in the State Vector. If the aircraft has crossed the altitude or speed threshold, then that aircraft is removed from the X-list or EX-list and entered into the other list using the technique described in the Initial Entry of Aircraft Into the X-list, EX-list, and Sector List (Section 6.1.3). If the aircraft has not crossed the thresholds, then its position in either list is updated according to its present x coordinate. At the same time, the aircraft position in the ATARS Sector List is updated to correspond to the present x coordinate. If the new position in the sector list is the first position, the executive program must be notified to correctly set the start pointer for the sector list.

6.3 Terrain/Airspace/Obstacle Avoidance

The capability to provide an alert for close proximity to the terrain, the violation of restricted airspace, and close proximity to an obstacle is provided by this task. The task operates on the linked list of aircraft for an ATARS sector. The logic to determine the need for an alert and send a Controller Alert Message, if required, is provided here; the actual construction of alert messages for uplink to the aircraft is performed by the Data Link Message Construction Task. Only aircraft which are receiving ATARS service or which are controlled are eligible for processing. The format of the Controller Alert Message for terrain/airspace/obstacle alerts is the same as shown in Table 11-1, with the following exceptions: (1) AMTYP is set to 01, 11, or 10 to indicate a terrain, restricted airspace, or obstacle alert, respectively, (2) only one aircraft is identified, and (3) the V1 field is used to indicate that the aircraft is unequipped with ATARS, implying the need for controller voice communication to convey the alert to the aircraft.

6.3.1 Terrain Avoidance Processing

The terrain avoidance logic makes use of the mode C barometric pressure correction provided with the surveillance report. Aircraft which are on final approach or which are taking off or landing at an airport may be below the terrain altitude threshold momentarily. Therefore, special checks are required to inhibit alerts if aircraft are on the final approach glide slope or operating in close proximity to an airfield.

The real-time processing of the map of the terrain is similar to the method used in the Terminal Area Minimum Safe Altitude Warning (MSAW) function described in Reference 7. A major difference concerns the use of a grid map with variable bin sizes and the associated indexing required to access the altitude threshold for each high-level and sub-level terrain bin. Terrain map processing includes the logic to access the terrain map, project the aircraft horizontally, determine the bin altitude thresholds along the projected path, and compare the thresholds to the projected aircraft altitude. Figure 6-4 gives an example of the bins to be checked. Table 6-1 summarizes the altitude checks required for each event in the projection of the aircraft. Terrain map processing also saves the altitude threshold for the initial bin in the aircraft State Vector, for potential use later as a constraint in the selection of resolution advisories for a conflict.

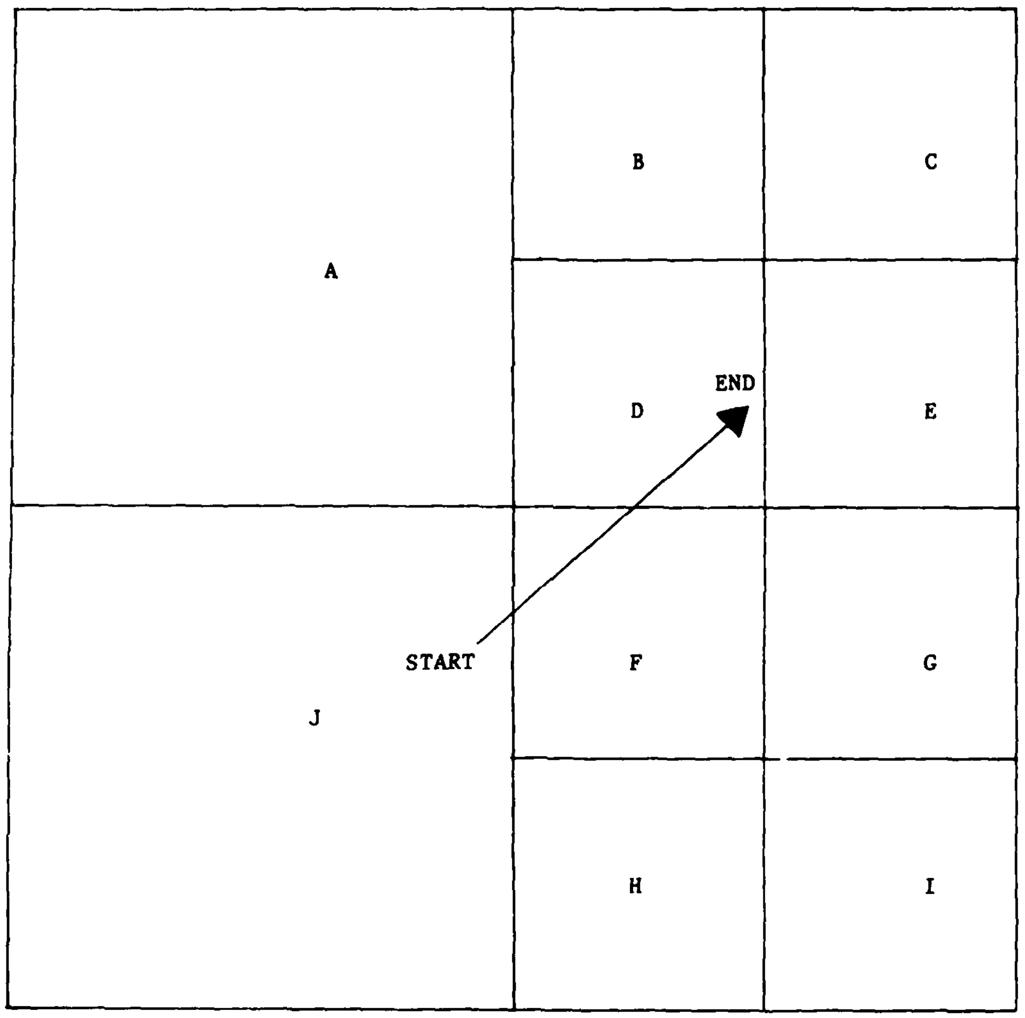
The off-line generation of the terrain map is described in Figure 6-5. This procedure is identical to the method used in Reference 7, with the exception of the sub-level indexing and the marking of bins which are in a final approach zone. A data structure implementation for the terrain map is presented in Figure 6-6.

6.3.2 Obstacle Avoidance Processing

The obstacle avoidance logic is only applied to aircraft which are below a minimum altitude. An x ordered linked list of obstacles is generated off-line and stored for access by obstacle avoidance processing. This list contains position and altitude information for each obstacle. Proximity to an obstacle is determined by applying fixed x, y search limits to the current position data and comparing the pressure-corrected altitude to the altitude of the obstacle. A check for convergence with the obstacle is made before issuing an alert. Multiple alert messages may be uplinked to the aircraft if convergence with more than one obstacle is detected.

6.3.3 Restricted Airspace Avoidance Processing

The restricted airspace avoidance logic consists of two major elements: (1) providing an advisory to uncontrolled aircraft upon first entry into a Terminal Control Area (TCA), and (2) providing an alert to any aircraft which has entered an area of restricted airspace. The technique for storage and access of the TCA map should allow effective use of the ATARS processors. The storage of the restricted airspace areas and the logical



ALTITUDE THRESHOLDS WHICH ARE CHECKED:
J, A, F, D, E. (NO CHECK ON G BECAUSE
END POINT TOO FAR AWAY.)

FIGURE 6-4
BIN CHECK EXAMPLE

TABLE 6-1

TERRAIN ALTITUDE CHECKS

<u>Start Point of Projection</u>

1. Current bin z threshold
2. Adjacent bin(s) z threshold

<u>X-line or Y-line Cross Point</u>

1. If negative vertical rate: z threshold of previous bin (use aircraft z at cross point)
2. If non-negative vertical rate: new bin z threshold
3. Adjacent bin(s) z threshold

<u>End Point of Projection</u>

1. If negative vertical rate: current bin z threshold
2. Adjacent bin(s) z threshold

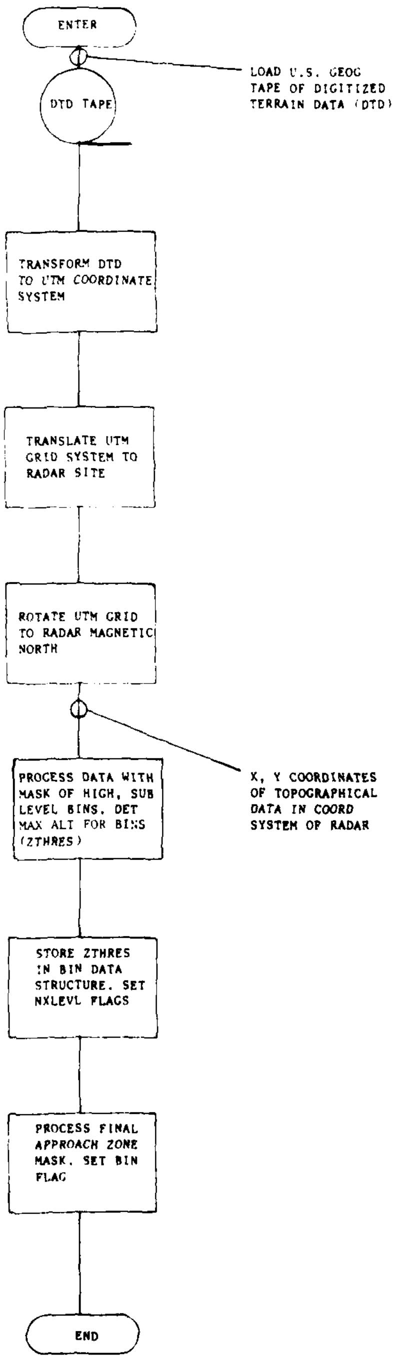

ENTER
LOAD U.S. GEOG TAPE OF DIGITIZED TERRAIN DATA (DTD)
DTD TAPE
TRANSFORM DTD TO UTM COORDINATE SYSTEM
TRANSLATE UTM GRID SYSTEM TO RADAR SITE
ROTATE UTM GRID TO RADAR MAGNETIC NORTH
X, Y COORDINATES OF TOPOGRAPHICAL DATA IN COORD SYSTEM OF RADAR
PROCESS DATA WITH MASK OF HIGH, SUB LEVEL BINS. DET MAX ALT FOR BINS (ZTHRES)
STORE ZTHRES IN BIN DATA STRUCTURE. SET NXLEVL FLAGS
PROCESS FINAL APPROACH ZONE MASK. SET BIN FLAG
END

FIGURE 6-5
DESCRIPTION OF TERRAIN MAP GENERATION

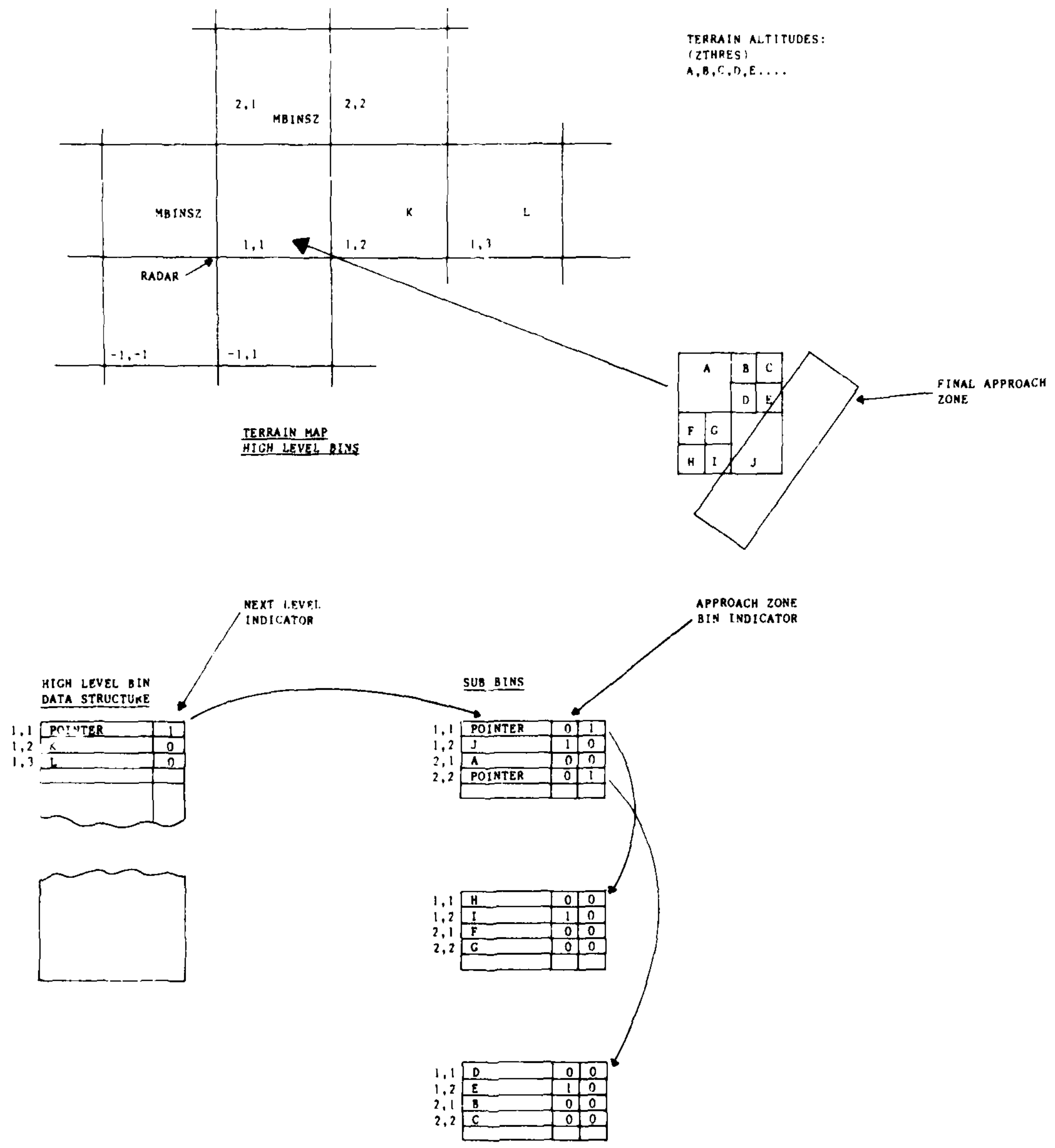


FIGURE 6-6
TERRAIN MAP DATA STRUCTURE

checks for determining if an aircraft is inside an area is the same as that described for processing of airport area types (see Section 8.7).

6.4 Pseudocode for Aircraft Processing

The pseudocode for New Aircraft Processing Task, Aircraft Update Processing Task, and Terrain/Airspace/Obstacle Avoidance Task follows.

PSEUDOCODE TABLE OF CONTENTS

PSEUDOCODE TABLE OF CONTENTS

```
-----------------------------------------------------------------------------------

      <*** THE VARIABLES LISTED BELOW ARE LOCAL TO THE NEW AIRCRAFT TASK ***>

STRUCTURE NEWACVBL

<*** POINTER VARIABLES ***>
  GROUP pointers
     PTR LSTPTR          < start pointer for various lists >
     PTR TMPTR2          < local (temporary) variable >

  GROUP signp
     INT NSIGNP          < lower integer value of signpost closest to aircraft >

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

TASK NEW_AIRCRAFT_PROCESSING
   IN (XINIT list for sector)
   INOUT (state vector);

     < Place new aircraft selected by tracker on x/ex_list >

     REPEAT WHILE (more AC on XINIT list);
          CALL X_EX_LIST_INITIAL_ENTRY;
          Initialize sector start pointers, final approach zone indicator, flags;
     ENDREPEAT;

END NEW_AIRCRAFT_PROCESSING;
```

------------------- NEW AIRCRAFT PROCESSING HIGH-LEVEL LOGIC -------------------

```
------------------------------------------------------------------------------------
TASK NEW_AIRCRAFT_PROCESSING
   IN (XINIT)
   INOUT (SVECT);

     < Place new aircraft selected by tracker on x/ex_list >

     REPEAT WHILE (more AC on XINIT list);
          CALL X_EX_LIST_INITIAL_ENTRY
               IN (CSCREEN, SYSTEM)
               INOUT (x/ex_list, SVECT, antenna sector list, SIDSPX, SIDSPE);
          SVECT.ACNES = $NULL:
          SVECT.PWPTR = $NULL;
          SVECT.UPNES = $NULL;
          SVECT.FAZ = $UDFAZ;
          SVECT.ATSEQ = $UNEQ;
          CLEAR SVECT.CENTR;
          CLEAR SVECT.XUPFL;
          CLEAR SVECT.HUBFLG;
          SET SVECT.INIFL;
     ENDREPEAT;

END NEW_AIRCRAFT_PROCESSING;
```

```
ROUTINE X_EX_LIST_INITIAL_ENTRY
   IN (coarse screen parameters, system parameters)
   INOUT (x/ex_list, state vector, antenna sector list, sector start pointers);

     < Place aircraft initially on x/ex_list >

     Calculate horizontal position and speed;

     IF (AC within hub range)
          THEN SET hub flag;
          ELSE CLEAR hub flag;

     Find nearest signpost to location of aircraft in the x plane;
     < Signpost located to the left of AC with positive x and to the right
        of AC with negative x >
     IF ((non_mode C report AND high speed) OR (mode C report AND
               (high alt OR high speed)))
          THEN Find corresponding signpost on ex_list;
               Find start of sector ex_list;
          ELSE Find corresponding signpost on x_list;
               Find start of sector x_list;

     PERFORM x_ex_list_placement;
     PERFORM sector_list_placement;

END X_EX_LIST_INITIAL_ENTRY;
```

```
--------------------------------------------------------------------------------
ROUTINE X_EX_LIST_INITIAL_ENTRY
   IN (CSCREEN, SYSTEM)
   INOUT (x/ex_list, SVECT, antenna sector list, SIDSPX, SIDSPE);

     < Place aircraft initially on x/ex_list >

     SVECT.X = SVECT.XP;
     SVECT.Y = SVECT.YP;
     SVECT.VSQ = SVECT.XDE**2 + SVECT.YDE**2;

     IF (SVECT.RHOP LT SYSTEM.HUBRAD)
          THEN SET SVECT.HUBFLG;
          ELSE CLEAR SVECT.HUBFLG;

     NSIGNP = INT( SVECT.X / CSCREEN.XSP );

     IF (( SVECT.MCFLG EQ $FALSE AND SVECT.VSQ GT SYSTEM.SPLO2 )
                OR ( SVECT.MCFLG EQ $TRUE AND (SVECT.Z GT SYSTEM.ALO OR
                 SVECT.VSQ GT SYSTEM.SPLO2 )))
          THEN SET SVECT.EXFLG;
                Use NSIGNP to get into ex_list array;
                Find start of sector ex_list from SIDSPE;
          ELSE CLEAR SVECT.EXFLG;
                Use NSIGNP to get into x_list array;
                Find start of sector x_list from SIDSPX;

     PERFORM x_ex_list_placement;
     PERFORM sector_list_placement;

END X_EX_LIST_INITIAL_ENTRY;
```

```
--------------------------------------------------------------------------------------

PROCESS sector_list_placement;

     < Place aircraft on antenna sector list >

     Point to start of sector list;
     IF (sector list is empty)
          THEN Record AC as start of sector list;

          ELSE REPEAT WHILE (more entries AND AC x_coord GT
                              list entry x_coord);
                         <step through list in forward direction>
                    Save pointer to previous entry;
                    Find next entry forward on list;
               ENDREPEAT;

               IF (no more entries on sector list)
                    THEN Link AC after last entry found; <current entry>
                    ELSE Link AC behind current entry;
                         IF (no previous entry saved) <loop never executed>
                              THEN record AC as start of sector list;

END sector_list_placement;
```

```
--------------------------------------------------------------------------------
PROCESS sector_list_placement;

     < Place aircraft on antenna sector list >

     LSTPTR = start of sector list from SIDSPX or SIDSPE;
     TMPTR2 = $NULL;

     IF (LSTPTR EQ $NULL)
          THEN Record subject AC as start of sector list;

          ELSE REPEAT WHILE ((SVECT.NEXTA NE $NULL) AND (SVECT.X GT SVECT(LSTPTR).X));
                    TMPTR2 = LSTPTR;
                    LSTPTR = SVECT.NEXTA;
               ENDREPEAT;

               IF (SVECT.NEXTA EQ $NULL)
                    THEN Link SVECT.NEXTA of last entry found (LSTPTR) to subject AC;
                    ELSE Link subject AC's SVECT.NEXTA TO LSTPTP AC;
                         IF (TMPTR2 EQ $NULL)    <loop never executed>
                              THEN Record AC as start of sector list SIDSPE
                                   or SIDSPX;
END sector_list_placement;
```

```
------------------------------------------------------------------------------------------
PROCESS x_ex_list_placement;

     < Link aircraft into x/ex_list >

     Point to signpost;
     IF (AC x_coord is positive)
          THEN REPEAT WHILE (more entries AND AC x_coord GT
                                     list entry x_coord);
                    Find next entry forward on list;
               ENDREPEAT;

               IF (no more entries on list)
                    THEN Link AC ahead of last entry found;
                    ELSE Link AC behind current entry;

          ELSE REPEAT WHILE (more entries AND AC x_coord LT
                                     list entry x_coord);
                    Find next entry behind on list;
               ENDREPEAT;

               IF (no more entries on list)
                    THEN Link AC behind last entry found;
                    ELSE Link AC ahead of current entry;

END x_ex_list_placement;
```

```
--------------------------------------------------------------------------------

PROCESS x_ex_list_placement;

     < Link aircraft into x/ex_list >

     LSTPTR = NSIGNP;
     IF (SVECT.X GT 0 )
          THEN REPEAT WHILE ((NEXTX NE $NULL) AND (SVECT.X GT SVECT(LSTPTR).X));
                    LSTPTR = SVECT.NEXTX;
               ENDREPEAT;

               IF (LSTPTR EQ $NULL)
                    THEN Link subject AC ahead of last AC on list;
                    ELSE Link subject AC behind next AC on list;

          ELSE REPEAT WHILE ((SVECT.PREVX NE $NULL) AND (SVECT.X LT SVECT(LSTPTR).X));
                    LSTPTR = SVECT.PREVX;
               ENDREPEAT;

               IF (LSTPTR EQ $NULL)
                    THEN Link subject AC behind last AC on list;
                    ELSE Link subject AC ahead of next AC on list;

END x_ex_list_placement;
```

```
--------------------------------------------------------------------------------

<*** THE VARIABLES LISTED BELOW ARE LOCAL TO THE AIRCRAFT UPDATE PROCESSING TASK ***>

STRUCTURE ACUPVBL

<*** POINTER VARIABLES ***>
  GROUP pointers
     PTR TEMPTR          < start pointer for various lists >
     PTR TMPTR2          < local (temporary) variable >

<*** TIME VARIABLES ***>
  GROUP times
     FLT DT              < time difference of predicted and present quarter cycle >
     FLT DTL             < level position correction time delta >
     FLT TATSN           < time of ATARS quarter cycle >
     FLT TEN             < sector time >

  GROUP flags
     BIT INZONE          < AC in map zone >

ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------
TASK AIRCRAFT_UPDATE_PROCESSING
   IN (sector start pointers, system variables, system parameters,
   OUT (conflict tables, pair records, messages)
   INOUT (x/ex_list, state vector);

     < Update aircraft position and hub update >
     Initialize sector time;
     REPEAT UNTIL (x and ex sector lists processed);
          REPEAT WHILE (more AC on sector list);
               IF (this AC not updated already)
                    THEN Initialize final approach zone indicator and AC area type;
                         CALL HORIZONTAL_COMPUTE;
                         SET AC update flag;  <XUPFL>
                         IF (non mode C reported AC)
                              THEN;  <omit vertical position, velocity computation>
                              ELSE Update vertical position and velocity;
                         IF (ATARS svc flag set) <by geog_proc last scan or track
                                        update this scan>
                              THEN PERFORM geographical_processing;
                              ELSE;
               PERFORM x_ex_list_updating;

          ENDREPEAT;

     ENDREPEAT;

     IF (this ATARS sector starts new quarter cycle)
          THEN PERFORM hub_update;
          ELSE; <skip hub update>

END AIRCRAFT_UPDATE_PROCESSING;


------------------ AIRCRAFT UPDATE PROCESSING HIGH-LEVEL LOGIC -------------------
```

```
TASK AIRCRAFT_UPDATE_PROCESSING
   IN (SIDSPX, SIDSPE, SYSVAR, SYSTEM)
   OUT (CTENTRY, PREC, messages)
   INOUT (x/ex_list, SVECT);

     < Update aircraft position and hub update >
     TEN = sector time;
     REPEAT UNTIL (x and ex sector lists processed);
          REPEAT WHILE (more AC on sector list);
               IF (SVECT.XUPFL EQ $FALSE)
                    THEN SVECT.FAZ = $UDFAZ;
                         SVECT.ACAT = $UNAT;
                         SVECT.XPRJ, SVECT.YPRJ, SVECT.ZPRJ = $UDPOS;
                         SVECT.XDPRJ, SVECT.YDPRJ, SVECT.ZDPRJ = $UDVEL;
                         DTL = TEN - SVECT.TMP;
                         CALL HORIZONTAL_COMPUTE
                              IN (DTL)
                              INOUT (SVECT);
                         SET SVECT.XUPFL;
                         IF (SVECT.MCFLG EQ $FALSE)
                              THEN; <omit vertical position, velocity computation>
                              ELSE SVECT.Z = SVECT.ZS + SVECT.ZDE *
                                                  (SYSVAR.CTIME - SVECT.TLUPD);
                                   SVECT.ZD = SVECT.ZDE;
                         IF (SVECT.ATSS EQ $TRUE)
                              THEN PERFORM geographical_processing;
               PERFORM x_ex_list_updating;
          ENDREPEAT;
     ENDREPEAT;
     IF ( (N/4) EQ (1. OR 2. OR 3. OR 4.) )
          < where N = sector numbers 1 thru 16 >
          THEN TATSN =SYSVAR.CTIME;
               PERFORM hub_update;
          ELSE; <skip hub update>
END AIRCRAFT_UPDATE_PROCESSING;
```

------------------- AIRCRAFT UPDATE PROCESSING LOW-LEVEL LOGIC -------------------

```
PROCESS geographical_processing;

     <Determine if aircraft in ATARS service area. Determine BCAS and
      squitter control states. Send Handoff Messages to other sites for
      aircraft leaving service area.>

     IF (Backup-Master mode)
          THEN PERFORM backup_master_updating;
                    <check Center zone and backup service areas separately>
          ELSE CALL AREA_MAP_TEST;
                    <for ATARS service area map in use for normal or backup status>
               IF (in service area AND own site ID not set in GEOG)
                    THEN SET own ID bit in GEOG; <just entered service area>
                         CLEAR Remote flag in conflict table entry (if any);
                         IF (AC DABS equipped)
                              THEN Send START ATARS SERVICE message to DABS;

     IF (in service area)
          THEN PERFORM bcas_level_control;
          ELSE IF (squitter lockout set)
                    THEN Send STOP SQUITTER LOCKOUT message to DABS;
               IF (AC DABS equipped AND own ID bit set in GEOG)
                              <just left service area>
                    THEN Send STOP ATARS SERVICE message to DABS;
               CLEAR own ID bit in GEOG;
               SET drop ATARS service flag;
               CLEAR ATARS service flag;
               IF (AC in conflict table AND any connected site(s) see AC)
                    THEN SET handoff bit in all pair records for AC
                              controlled by own site;
                         Send HANDOFF messages for such pairs to connected
                              sites that see this AC;

END geographical_processing;
```

------------------ AIRCRAFT UPDATE PROCESSING HIGH-LEVEL LOGIC ------------------

```
--------------------------------------------------------------------------------

PROCESS geographical_processing;

     <Determine if aircraft in ATARS service area. Determine BCAS and
       squitter control states. Send Handoff Messages to other sites for
       aircraft leaving service area.>

     IF (SYSVAR.MASTER EQ $TRUE)
          THEN PERFORM backup_master_updating;
          ELSE CALL AREA_MAP_TEST
                    IN (SYSVAR.MAPPTR, SVECT)
                    OUT (INZONE);
               IF (INZONE EQ $TRUE AND SYSTEM.OWNID bit not set in SVECT.GEOG;
                    THEN SET SYSTEM.OWNID bit in SVECT.GEOG;
                         IF (SVECT.CTE not null)
                              THEN CLEAR CTENTRY.REMFLG;
                         IF (SVECT.ATIFLG EQ $FALSE)  <DABS aircraft>
                              THEN Send START ATARS SERVICE message to DABS;

     IF (INZONE EQ $TRUE)
          THEN PERFORM bcas_level_control;
          ELSE IF (SVECT.SQLO set)
                    THEN Send STOP SQUITTER LOCKOUT Message to DABS;
               IF (SVECT.ATIFLG EQ $FALSE AND SYSTEM.OWNID bit set in SVECT.GEOG)
                    THEN Send STOP ATARS SERVICE Message to DABS;
               CLEAR SYSTEM.OWNID bit in SVECT.GEOG;
               SET SVECT.DRATS;
               CLEAR SVECT.ATSS;
               IF (SVECT.CTPTR not null AND any connected site(s) in SVECT.GEOG)
                    THEN SET PREC.HDOFF in all pair records for AC
                                             with PREC.ATSID=OWNID;
                         Send HANDOFF msgs for such pairs to connected
                                             sites in SVECT.GEOG;

END geographical_processing;
```

```
--------------------------------------------------------------------------------
PROCESS x_ex_list_updating;

     < Update x position on x/ex_list >

     IF (AC on ex_list AND ((mode C report AND (low alt AND
          low speed)) OR (non_mode C report AND low speed)))
          THEN Remove AC from ex_list;
               CALL X_EX_LIST_INITIAL_ENTRY; <add aircraft to x_list>

     ELSEIF (AC on x_list AND ((mode C report AND (high alt OR
               highspeed)) OR (non_mode C report AND high speed)))
          THEN Remove AC from x_list;
               CALL X_EX_LIST_INITIAL_ENTRY; <add aircraft to ex_list>

     OTHERWISE PERFORM x_ex_list_relink;
               PERFORM sector_thread_update;

END x_ex_list_updating;
```

```
------------------------------------------------------------------------

PROCESS x_ex_list_updating;

     < Update x position on x/ex_list >

     IF (SVECT.EXFLG EQ $TRUE AND ((SVECT.MCFLG EQ $TRUE AND
          (SVECT.Z LE SYSTEM.ALO AND SVECT.VSQ LE SYSTEM.SPLO2)) OR
          (SVECT.MCFLG EQ $FALSE AND SVECT.VSQ LE SYSTEM.SPLO2)))
          THEN Remove AC from ex_list;
               CALL X_EX_LIST_INITIAL_ENTRY  <add aircraft to x_list>
                  IN (CSCREEN, SYSTEM)
                  INOUT (x/ex_list, SVECT, SIDSPX, SIDSPE, antenna sector list);

     ELSEIF (SVECT.EXFLG EQ $FALSE AND ((SVECT.MCFLG EQ $TRUE AND
               (SVECT.Z GT SYSTEM.ALO OR SVECT.VSQ GT SYSTEM.SPLO2)) OR
                (SVECT.MCFLG EQ $FALSE AND SVECT.VSQ GT SYSTEM.SPLO2)))
          THEN Remove AC from x_list;
               CALL X_EX_LIST_INITIAL_ENTRY  <add aircraft to ex_list>
                    IN (CSCREEN, SYSTEM)
                    INOUT (x/ex_list, SVECT, SIDSPX, SIDSPE, antenna sector list);

     OTHERWISE PERFORM x_ex_list_relink;
               PERFORM sector_thread_update;

END x_ex_list_updating;
```

```
--------------------------------------------------------------------------------
PROCESS hub_update;

     < Recompute position for hub aircraft >

     REPEAT UNTIL (x and ex lists processed);
          Find signpost with x_value LE negative of hub radius;
          <move up list in positive x direction>
          REPEAT UNTIL (end of list OR signpost reached with x_value
                              GE positive hub radius);
               IF (next entry is an AC <not a signpost> AND hub flag
                              is set AND AC not yet updated)
                    THEN Update horizontal position, velocity coordinates;
                         SET AC update flag;  <XUPFL>
                         IF (non_mode C reported AC)
                              THEN;  <omit vertical position, velocity computation>
                              ELSE Update vertical position, vertical velocity;
                         PERFORM x_ex_list_updating;
          ENDREPEAT;

     ENDREPEAT;

     REPEAT UNTIL (x and ex lists processed);

          Find signpost with x_value LE negative of hub radius;

          REPEAT UNTIL (end of list OR signpost reached with x_value
                              GE positive hub radius);
               IF (next entry is an AC AND hub flag set)
                    THEN CLEAR update flag;
          ENDREPEAT;

     ENDREPEAT;

END hub_update;

-----------------  AIRCRAFT UPDATE PROCESSING HIGH-LEVEL LOGIC  ------------------
```

```
--------------------------------------------------------------------------------
PROCESS hub_update;

     < Recompute position for hub aircraft >

     REPEAT UNTIL (x and ex lists processed);
          Find signpost with (SVECT.NEXTX) LE (-SYSTEM.HUBRAD);
          < move up list in positive x direction >
          REPEAT UNTIL (end of list OR signpost reached with SVECT.NEXTX
                         GE (+SYSTEM.HUBRAD);
               IF ((SVECT.SPIDFG EQ $FALSE) AND (SVECT.HUBFLG EQ $TRUE) AND
                         (SVECT.XUPFL EQ $FALSE))
                    THEN DT = TATSN - SVECT.TMP;
                         CALL HORIZONTAL_COMPUTE
                              IN (DT)
                              INOUT (SVECT);
                         SET SVECT.XUPFL;
                         IF (SVECT.MCFLG EQ $FALSE)
                              THEN;  <omit SVECT.Z, SVECT.ZD computation>
                              ELSE SVECT.Z = SVECT.ZS + SVECT.ZDE *
                                             (SYSVAR.CTIME - SVECT.TLUPD)
                                   SVECT.ZD = SVECT.ZDE;
                         PERFORM x_ex_list_updating;
          ENDREPEAT;
     ENDREPEAT;
     REPEAT UNTIL (x and ex lists processed);

          Find signpost with SVECT.NEXTX LE (-SYSTEM.HUBRAD);
          REPEAT UNTIL (end of list OR signpost reached with SVECT.NEXTX
                         GE (+SYSTEM.HUBRAD);
               IF ((SVECT.SPIDFG EQ $FALSE) AND (SVECT.HUBFLG EQ $TRUE))
                    THEN CLEAR SVECT.XUPFL;
          ENDREPEAT;
     ENDREPEAT;
END hub_update;

------------------- AIRCRAFT UPDATE PROCESSING LOW-LEVEL LOGIC -------------------
```

```
--------------------------------------------------------------------------------
PROCESS backup_master_updating;

   <Determine whether AC is in backup-service zone, center zone, or outside service.>

     CALL AREA_MAP_TEST;   <for Center zone>

     IF (in center zone)
          THEN SET FAILED site's ID in GEOG;
               CLEAR Remote flag in conflict table entry (if any);
               IF (AC not already marked for CENTER status AND no conflicts
                              exist with own_ID in charge)
                    THEN Send START ATARS SERVICE msg to DABS with FAILED site ID;
                         SET CENTER status flag for AC;

          ELSE CALL AREA_MAP_TEST;   <for backup service map>
               IF (aircraft in backup service zone)
                    THEN SET own_ID in GEOG;
                         CLEAR FAILED site's ID in GEOG;
                         CLEAR Remote flag in conflict table entry (if any);
                         IF ((AC marked for CENTER status OR own ID not set in GEOG)
                AND DABS equipped AND no conflicts exist with FAILED site in charge)
                              THEN Send START ATARS SERVICE msg to DABS with own_ID;
                                   CLEAR CENTER status flag for AC;

                    ELSE AC not in service area;
                         IF (DABS equipped AND FAILED site ID set in GEOG)
                              <just left Center zone>
                              THEN Send STOP ATARS SERVICE message to DABS;
                                   CLEAR FAILED site ID bit in GEOG;

END backup_master_updating;
```

```
--------------------------------------------------------------------------------

PROCESS backup_master_updating;

   <Determine whether AC is in backup-service zone, center zone, or outside service>

      CALL AREA_MAP_TEST   <Test center zone map corresponding to FAILED site>
         IN (SYSVAR.CTRPTR, SVECT)
         OUT (INZONE);
      IF (INZONE EQ $TRUE)
         THEN SET failed site's ID in SVECT.GEOG;
              IF (SVECT.CTE not null)
                 THEN CLEAR CTENTRY.REMFLG;
              IF (SVECT.CENTR not set AND no pair records with PREC.ATSID=OWNID)
                 THEN Send START ATARS SERVICE msg to DABS with failed site ID;
                      SET SVECT.CENTR;

         ELSE <Test backup service map>
              CALL AREA_MAP_TEST
                 IN (SYSVAR.MAPPTR, SVECT)
                 OUT (INZONE);
              IF (INZONE EQ $TRUE)
                 THEN SET SYSTEM.OWNID bit in SVECT.GEOG;
                      CLEAR failed site's ID in SVECT.GEOG;
                      IF (SVECT.CTE not null)
                         THEN CLEAR CTENTRY.REMFLG;
                      IF ((SVECT.CENTR EQ $TRUE OR SYSTEM.OWNID bit not set in
   SVECT.GEOG) AND SVECT.ATIFLG EQ $FALSE AND no pr recs. with PREC.ATSID=failed site)
                         THEN Send START ATARS SVC msg with SYSTEM.OWNID;
                              CLEAR SVECT.CENTR;

                 ELSE <AC not in service area>
                      IF (SVECT.ATIFLG EQ $FALSE AND failed site ID in SVECT.GEOG)
                         THEN Send STOP ATARS SERVICE message to DABS;
                              CLEAR failed site ID bit in SVECT.GEOG;

END backup_master_updating;
------------------- AIRCRAFT UPDATE PROCESSING LOW-LEVEL LOGIC -------------------
```

```
--------------------------------------------------------------------------------------------------

PROCESS bcas_level_control;

          <Compute BCAS sensitivity level and DABS squitter lockout.>

     IF (AC equipped with BCAS)
          THEN Determine BCAS_sensitivity_level from map;
               IF (BCAS_sensitivity_level has changed)
                    THEN Send BCAS SLC message to DABS;

     IF (AC DABS-equipped)
          THEN Test position to see if now in squitter lockout area;
               IF (test shows status should change)
                    THEN Send START/STOP SQUITTER LOCKOUT message to DABS;

END bcas_level_control;
```

```
--------------------------------------------------------------------------------

PROCESS bcas_level_control;

          <compute BCAS sensitivity level and DABS squitter lockout>

     IF (SVECT.ATSEQ EQ $BCAS)
          THEN Determine BCAS_SENSITIVITY_LEVEL from map;
               IF (BCAS_SENSITIVITY_LEVEL NE SVECT.BCASSL)
                    THEN Send BCAS SLC msg to DABS specifying
                          new BCAS_SENSITIVITY_LEVEL;
                         SVECT.BCASSL=BCAS_SENSITIVITY_LEVEL;

     IF (SVECT.ATIFLG  EQ $FALSE)  <DABS aircraft>
          THEN <Test position to see if now in squitter lockout area>
               CALL AREA_MAP_TEST
                  IN (SYSTEM.SQMAP, SVECT)
                  OUT (INZONE):
               IF (INZONE NE SVECT.SQLO)
                    THEN Send START/STOP SQUITTER LOCKOUT msg to DABS
                          according to INZONE;
                         <INZONE=$TRUE denotes START>
                         SVECT.SQLO=INZONE;

END bcas_level_control;
```

```
-------------------------------------------------------------------------------------

PROCESS sector_thread_update;

     < Update aircraft position on ATARS sector thread list >

     IF (AC on ex_list)
          THEN Find start of sector ex_list;
          ELSE Find start of sector x_list;

     REPEAT WHILE (more entries on sector list AND subj AC x_coord
                         GT list entry x_coord);
          Find next entry on list;
          IF (subject AC previous entry)
               THEN Delete previous entry from list;
     ENDREPEAT;

     IF (next list entry is subj AC)
          THEN;
     ELSEIF (no more entries)
          THEN Link subject AC after last entry found; <current entry>
     OTHERWISE Link subj AC before current entry on list;
               IF (no previous entry saved) <loop never executed>
                    THEN Record subj AC as start of sector list;

END sector_thread_update;
```

```
--------------------------------------------------------------------------------
PROCESS sector_thread_update;

     < Update aircraft position on ATARS sector thread list >

     TMPTR2 = $NULL;

     IF (SVECT.EXFLG EQ $TRUE)
          THEN Find start (TEMPTR) of sector ex_list from SIDSPE;
          ELSE Find start (TEMPTR) of sector x_list FROM SIDSPX;

     REPEAT WHILE ((SVECT.NEXTA NE $NULL) AND (SVECT.X GT SVECT(TEMPTR).X));
          TMPTR2 = TEMPTR;
          TEMPTR = SVECT.NEXTA;
          IF (subject AC previous entry)
               THEN Delete previous entry from list;
     ENDREPEAT;

     IF (next list entry is subject AC)
          THEN;
     ELSEIF (SVECT.NEXTA EQ $NULL)
          THEN Link SVECT.NEXTA of last entry found (TEMPTR) to subject aircraft;
     OTHERWISE Link subject AC's SVECT.NEXTA to TEMPTR AC;
               IF (TMPTR2 EQ $NULL) <loop never executed>
                    THEN Record subject AC as start of sector list SIDSPE or SIDSPX;

END sector_thread_update;
```

```
--------------------------------------------------------------------------------
PROCESS x_ex_list_relink;

    < Process x/ex_list >

    Find x_coordinate of aircraft next on list after subject AC;
    IF (next entry signpost AND subj AC x_coord GT next AC x_coord)
        THEN REPEAT WHILE (more entries on list AND subj AC x_coord
                          GT next AC x_coord);
                 Find x_coord of next AC on list;
             ENDREPEAT;

             Remove subj AC from old position;
             IF (no more entries on list)
                 THEN Link subj AC after last AC on list;
                 ELSE Link subj AC behind current AC on list;

        ELSE Find x_coord of AC next on list behind subj AC;
             IF (next entry signpost AND subj AC x_coord LT next AC x_coord)
                 THEN REPEAT WHILE (more entries on list AND subj AC x_coord
                                   LT next AC x_coord);
                          Find x_coord of next AC on list searching backwards;
                      ENDREPEAT;

                      Remove subj AC from old position;
                      IF (no more entries on list)
                          THEN Link subj AC behind last AC found;
                          ELSE Link subj AC ahead of current AC on list;
                 ELSE;

END x_ex_list_relink;
```

```
--------------------------------------------------------------------------------
PROCESS x_ex_list_relink;

     < Process x/ex_list >

     TEMPTR = SVECT.NEXTX;

     IF ((TEMPTR NE $NULL) AND (SVECT.X GT SVECT(TEMPTR).X))
          THEN REPEAT WHILE ((TEMPTR NE $NULL) AND (SVECT.X GT SVECT(TEMPTR).X));
                    TEMPTR = SVECT.NEXTX;
               ENDREPEAT;

               Remove subj AC from old position;
               IF (TEMPTR EQ $NULL)
                    THEN Link subj AC after last AC on list;
                    ELSE Link subj AC behind current AC on list;

          ELSE TEMPTR = SVECT.PREVX;
               IF ((TEMPTR NE $NULL) AND (SVECT.X LT SVECT(TEMPTR).X))
                    THEN REPEAT WHILE ((TEMPTR NE $NULL) AND (SVECT.X LT
                                                            SVECT(TEMPTR).X));
                              TEMPTR = SVECT.PREVX;
                         ENDREPEAT;

                         Remove subj AC from old position;
                         IF (TEMPTR EQ $NULL)
                              THEN Link subj AC behind last AC on list;
                              ELSE Link subj AC ahead of current AC on list;
                    ELSE;

END x_ex_list_relink;
```

```
--------------------------------------------------------------------------------
ROUTINE AREA_MAP_TEST
   IN (map pointer, state vector)
   OUT (indication that position is within area tested);

     IF (aircraft high altitude)
          THEN Select set of high altitude map vertices;
          ELSE Select low altitude map vertices;  <includes non-mode C AC>

     Position initially assumed to be within area;
     REPEAT UNTIL (all map vertices processed);
          Test next map boundary;
          IF (AC not within map zone)
               THEN Indicate position not within area tested;
     EXITIF (Position has been found outside area tested);
     ENDREPEAT;

END AREA_MAP_TEST;
```

```
ROUTINE AREA_MAP_TEST
   IN (MAPPTR, SVECT)
   OUT (INZONE);

     INT (NVERT, I);
     FLT (DX(NVERT), DY(NVERT), K(NVERT) );

     IF (SVECT.EXFLG EQ $TRUE)
          THEN Select set of high altitude map vertices;
          ELSE Select set of low altitude map vertices;
     INZONE=$TRUE;

     REPEAT UNTIL (I EQ NVERT);  <all map vertices processed>


     <DX(1,...,NVERT), DY(1,...,NVERT), and K(1,...,NVERT) are prestored
      constants corresponding to
      DX(I)=X(I+1)-X(I) (difference of adjacent map vertices' x-coordinates);
      DY(I)=Y(I+1)-Y(I) (difference of adjacent map vertices' y-coordinates);
      K(I)=Y(I)*DX(I) - X(I)*DY(I). Note when I=NVERT, X(I+1) means X(1).
      The following expression evaluates vector cross-products between
      the vectors connecting adjacent map vertices and those from each
      vertex to the AC position>


          IF ((SVECT.X * DY(I) - SVECT.Y * DX(I) +K(I) ) is negative)
               THEN INZONE=$FALSE;   <AC outside map area>
     EXITIF (INZONE EQ $FALSE);
     ENDREPEAT;

END AREA_MAP_TEST;
```

```
ROUTINE HORIZONTAL_COMPUTE
   IN (time difference)
   INOUT (state vector);

     Compute horizontal position and velocity components;

END HORIZONTAL_COMPUTE;
```

```
--------------------------------------------------------------------------------

ROUTINE HORIZONTAL_COMPUTE
   IN (DTL)
   INOUT (SVECT);

     SVECT.X = SVECT.XP + SVECT.XDE * DTL;
     SVECT.Y = SVECT.YP + SVECT.YDE * DTL;
     SVECT.XD = SVECT.XDE;
     SVECT.YD = SVECT.YDE;
     SVECT.VSQ = SVECT.XD**2 + SVECT.YD**2;

END HORIZONTAL_COMPUTE;
```

```
--------------------------------------------------------------------------------

<*** PARAMETERS USED IN TERRAIN/AIRSPACE/OBSTACLE AVOIDANCE ***>


STRUCTURE TAOPARM


  GROUP general_values

     FLT TRALT          < Altitude limit for terrain avoidance logic >

     FLT OBALT          < Altitude limit for obstacle avoidance logic >

     FLT OBXCK          < X-increment for obstacle proximity check >

     FLT OBYCK          < Y-increment for obstacle proximity check >

     FLT OBZCK          < Z-increment for obstacle proximity check >

     FLT TRHTM          < Horizontal projection time for terrain bin checks >


ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------

<*** VARIABLES USED IN TERRAIN/AIRSPACE/OBSTACLE AVOIDANCE ***>

STRUCTURE TAO

  GROUP misc_variables
    BIT TALRT            < Flag indicating terrain alert >
    BIT TVIOL            < Flag indicating terrain violation (potential alert) >
    BIT AZBIN            < Flag indicating approach zone bin >
    BIT OALRT            < Flag indicating obstacle alert >
    BIT TCALRT           < Flag indicating TCA alert >
    BIT RALRT            < Flag indicating restricted airspace alert >
    FLT ZMCC             < Pressure-corrected altitude of aircraft >

ENDSTRUCTURE;

STRUCTURE OBLIST

  GROUP obstacle_data
    INT NUMBER           < Obstacle number >
    FLT X                < X-coordinate of obstacle >
    FLT Y                < Y-coordinate of obstacle >
    FLT ALT              < Altitude of top of obstacle >
    PTR NEXTO            < Pointer to next obstacle in list >

ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------------

TASK TERRAIN_AIRSPACE_OBSTACLE_AVOIDANCE
   IN (Linked list of aircraft for a sector)
   OUT (PWILST entries, controller alert messages, and
        terrain altitude in state vector for each aircraft);

     < This task provides an alert for close proximity to the terrain, the violation
       of restricted airspace, and close proximity to an obstacle. >

     REPEAT UNTIL (there are no more aircraft on the sector list);

          Select next aircraft from sector list;

          IF (own site is providing ATARS service OR aircraft is controlled)
               THEN PERFORM terrain_avoidance;

          IF ((own site is providing ATARS service AND own site is primary) OR
              aircraft is controlled)
               THEN PERFORM obstacle_avoidance;
                    PERFORM restricted_airspace_avoidance;

     ENDREPEAT;

END TERRAIN_AIRSPACE_OBSTACLE_AVOIDANCE;
```

----------- TERRAIN/AIRSPACE/OBSTACLE AVOIDANCE TASK HIGH-LEVEL LOGIC -------------

```
--------------------------------------------------------------------------------

TASK TERRAIN_AIRSPACE_OBSTACLE_AVOIDANCE
   IN (State vector of first aircraft in ATARS sector list)
   OUT (PWILST entries, controller alert messages, and
        SVECT.TERALT for each aircraft);

     REPEAT UNTIL (SVECT.NEXTA EQ $NULL);

          IF (SVECT.ATSS EQ $TRUE OR SVECT.CUNC EQ $TRUE)
               THEN PERFORM terrain_avoidance;

          IF ((SVECT.ATSS EQ $TRUE AND SVECT.PSTAT EQ $TRUE) OR
              SVECT.CUNC EQ $TRUE)
               THEN PERFORM obstacle_avoidance;
                    PERFORM restricted_airspace_avoidance;

          Select next aircraft from sector list via SVECT.NEXTA;

     ENDREPEAT;

END TERRAIN_AIRSPACE_OBSTACLE_AVOIDANCE;
```

------------ TERRAIN/AIRSPACE/OBSTACLE AVOIDANCE TASK LOW-LEVEL LOGIC ------------

```
--------------------------------------------------------------------------------------

PROCESS terrain_avoidance;

    < This process determines the need for an alert for close proximity to the
      terrain. >

    IF (mode-C altitude LT maximum terrain altitude)
        THEN Pressure correct mode-C altitude;
            PERFORM terrain_map_processing;
            IF (there is a terrain violation)
                THEN IF (aircraft is in area type 1)
                        THEN :  < no terrain alert. >
                    ELSEIF (aircraft is in an approach zone bin)
                        THEN CALL FINAL_APPROACH_ZONE_DETERMINATION;
                                        < See Section 8. >
                            IF (aircraft is not in a final approach zone)
                                THEN generate terrain alert;
                                ELSE ;  < no terrain alert. >
                    OTHERWISE generate terrain alert;
                ELSE :  < no terrain alert. >
        ELSE store terrain altitude of zero in state vector;   < No terrain alert >

    IF (terrain alert generated)
        THEN IF (own site is providing ATARS service AND own site is primary)
                THEN CALL ENTER_ALERT_IN_PWILST;
                    Compute altitude relative to terrain and record
                      in PWILST entry;
            IF (aircraft is controlled)
                THEN CALL SEND_ALERT_TO_CONTROLLER;
        ELSE :  < take no action. >

END terrain_avoidance;
```

```
--------------------------------------------------------------------------------

PROCESS terrain_avoidance;

    CLEAR TALRT;

    IF (SVECT.Z LT TRALT)
        THEN Compute ZHCC by pressure-correcting SVECT.Z;
            PERFORM terrain_map_processing;
            IF (TVIOL EQ $TRUE)
                THEN IF (SVECT.ACAT EQ $AT1)
                        THEN :   < no terrain alert. >
                    ELSEIF (AZBIN EQ $TRUE)
                        THEN CALL FINAL_APPROACH_ZONE_DETERMINATION
                                  INOUT (SVECT);
                            IF (SVECT.FAZ EQ $FAZ0)
                                THEN SET TALRT;
                                ELSE :   < no terrain alert. >
                    OTHERWISE SET TALRT;
                ELSE :   < no terrain alert. >
        ELSE SVECT.TERALT = 0;

    IF (TALRT EQ $TRUE)
        THEN IF (SVECT.ATSS EQ $TRUE AND SVECT.PSTAT EQ $TRUE)
                THEN CALL ENTER_ALERT_IN_PWILST
                          IN (SVECT.PWPTR,'terrain alert',null identifier)
                          OUT (TERRAIN entry in PWILST);
                    TERRAIN.RELALT = ZHCC - SVECT.TERALT;
            IF (SVECT.CONC EQ $TRUE)
                THEN CALL SEND_ALERT_TO_CONTROLLER
                          IN (SVECT,'terrain alert')
                          OUT (Controller alert message);
        ELSE :   < take no action. >

END terrain_avoidance;
```

```
------------------------------------------------------------------------------------
PROCESS obstacle_avoidance;

     < This process determines the need for an alert for close proximity to an
       obstacle. >

     IF (mode-C altitude LT maximum obstacle altitude)
          THEN Pressure-correct mode-C altitude;
               Map aircraft position to X-ordered linked list of obstacles;
               Determine X, Y, Z search limits;
               IF (there are obstacles within the search limits)
                    THEN LOOP;   < Repeat for each such obstacle. >
                              IF (aircraft is converging with obstacle)
                                   THEN generate obstacle alert;
                         EXITIF (no more obstacles within search limits);
                         ENDLOOP;

     IF (any obstacle alerts generated)
          THEN IF (own site is providing ATARS service AND own site is primary)
                    THEN LOOP:   < Repeat for each obstacle alert. >
                              CALL ENTER_ALERT_IN_PWILST;
                              Compute range, clock bearing, and altitude relative
                                 to obstacle and record in PWILST entry;
                         EXITIF (alerts entered for all obstacles);
                         ENDLOOP;
               IF (aircraft is controlled)
                    THEN CALL SEND_ALERT_TO_CONTROLLER;
          ELSE ;   < take no action. >

END obstacle_avoidance;
```

```
--------------------------------------------------------------------------------
PROCESS obstacle_avoidance;

    FLT (DOT, RX, RY);

    CLEAR OALRT;
    IF (SVECT.Z LT OBALT)
        THEN Compute ZMCC by pressure-correcting SVECT.Z;
             Map aircraft position to X-ordered linked list of obstacles;
             Determine X, Y, Z search limits;
             IF (there are obstacles within the search limits)
                 THEN LOOP;   < Repeat for each such obstacle. >
                              DOT = -SVECT.XD * RX - SVECT.YD * RY;
                              IF (DOT LT 0)
                                  THEN SET OALRT;
                      EXITIF (no more obstacles within search limits);
                      ENDLOOP;

    IF (OALRT EQ $TRUE)
        THEN IF (SVECT.ATSS EQ $TRUE AND SVECT.PSTAT EQ $TRUE)
                 THEN LOOP;   < Repeat for each obstacle alert. >
                              CALL ENTER_ALERT_IN_PWILST
                                   IN (SVECT.PWPTR,'obstacle alert',OBLIST.NUMBER)
                                   OUT (OBSTACLE entry in PWILST);
                              OBSTACLE.RANGE = SQRT(RX**2 + RY**2);
                              Compute OBSTACLE.CLOCK_BRG as in proximity data calculation;
                              OBSTACLE.REL_ALT = ZMCC - OBLIST.ALT;
                      EXITIF (alerts entered for all obstacles);
                      ENDLOOP;
             IF (SVECT.CONC EQ $TRUE)
                 THEN CALL SEND_ALERT_TO_CONTROLLER
                          IN (SVECT,'obstacle alert')
                          OUT (Controller alert message);
        ELSE :   < take no action. >
END obstacle_avoidance;

------------ TERRAIN/AIRSPACE/OBSTACLE AVOIDANCE TASK LOW-LEVEL LOGIC ------------
```

```
--------------------------------------------------------------------------------------
PROCESS restricted_airspace_avoidance;

    < This process determines the need for an alert for an uncontrolled aircraft
      entering a TCA or an aircraft entering a restricted airspace. >

    < Check for TCA alert. >

    IF (aircraft is uncontrolled)
         THEN Apply coarse range and altitude TCA filter;
              IF (passed coarse TCA filter)
                   THEN Access TCA map;
                        Perform TCA position and altitude checks;
                        IF (aircraft is inside TCA)
                             THEN Generate TCA alert;
                                  CALL ENTER_ALERT_IN_PWILST;

    < Check for restricted airspace alert. >

    IF (TCA alert not generated)
         THEN Apply restricted airspace altitude filter;
              Search through list of restricted airspaces;
              IF (aircraft is in restricted area)
                   THEN Generate restricted airspace alert;
                        IF (own site is providing ATARS service AND
                            own site is primary)
                            THEN CALL ENTER_ALERT_IN_PWILST;
                        IF (aircraft is controlled)
                            THEN CALL SEND_ALERT_TO_CONTROLLER;
                   ELSE :   < take no action. >

END restricted_airspace_avoidance;
```

```
------------------------------------------------------------------------------------

PROCESS restricted_airspace_avoidance;

     CLEAR TCALRT;
     IF (SVECT.CUNC EQ $FALSE)
          THEN Apply coarse range and altitude TCA filter;
               IF (passed coarse TCA filter)
                    THEN Access TCA map;
                         Perform TCA position and altitude checks;
                         IF (aircraft is inside TCA)
                              THEN SET TCALRT;
                                   CALL ENTER_ALERT_IN_PWILST
                                           IN (SVECT.PWPTR,'TCA alert',null identifier)
                                           OUT (AIRSPACE entry in PWILST);

     CLEAR RALRT;
     IF (TCALRT EQ $FALSE)
          THEN Apply restricted airspace altitude filter;
               Search through list of restricted airspaces;
               IF (aircraft is in restricted area)
                    THEN SET RALRT;
                         IF (SVECT.ATSS EQ $TRUE AND SVECT.PSTAT EQ $TRUE)
                              THEN CALL ENTER_ALERT_IN_PWILST
                                           IN (SVECT.PWPTR,'restricted airspace alert',
                                               restricted airspace identifier)
                                           OUT (AIRSPACE entry in PWILST);
                         IF (SVECT.CUNC EQ $TRUE)
                              THEN CALL SEND_ALERT_TO_CONTROLLER
                                           IN (SVECT,'restricted airspace alert')
                                           OUT (Controller alert message);

END restricted_airspace_avoidance;
```

```
---------------------------------------------------------------------------
PROCESS adjacent_bin_checks;

     < This process checks aircraft altitude against the terrain thresholds of
       nearby terrain bins. >

     Determine if aircraft X,Y coordinates are within XBNER, YBNER
        limits of any adjacent bins;

     Compare aircraft altitude with thresholds of any such bins;

     IF (altitude LT any such threshold)
          THEN declare terrain violation;
          ELSE :   < no terrain violation. >

END adjacent_bin_checks;
```

```
--------------------------------------------------------------------------------
PROCESS adjacent_bin_checks;

     Determine if aircraft X,Y coordinates are within XBNER, YBNER
        limits of any adjacent bins;

     Compare ZMCC with thresholds of any such bins;

     IF (ZMCC LT any such threshold)
          THEN SET TVIOL;
          ELSE ;   < no terrain violation. >

END adjacent_bin_checks;
```

```
------------------------------------------------------------------------------
PROCESS terrain_map_processing:

     < This process predicts proximity to the terrain by accessing the terrain map
       and projecting the aircraft ahead. >

     Determine bin and altitude threshold using current X,Y coordinates;
     Store terrain altitude threshold in state vector;

     IF (aircraft is controlled OR own site is primary)

          THEN IF (altitude LT threshold)
                    THEN declare terrain violation;
                    ELSE PERFORM adjacent_bin_checks;
               IF (no terrain violation so far)
                    THEN < project aircraft ahead. >
                         Determine projection end point by projecting aircraft
                            straight ahead for TRHTM seconds;
                         LOOP;
                              Project aircraft to nearest X-crossing point,
                                 Y-crossing point, or end point;
                              IF (aircraft is descending AND
                                  new aircraft altitude LT old threshold)
                                   THEN declare terrain violation;
                                   ELSE PERFORM adjacent_bin_checks;
                         EXITIF (terrain violation OR end point reached);
                              Determine new bin and new threshold;
                              IF (aircraft is not descending AND
                                  aircraft altitude LT new threshold)
                                   THEN declare terrain violation;
                         EXITIF (terrain violation declared);
                         ENDLOOP;

          ELSE :   < no terrain violation. >

END terrain_map_processing;
```

```
--------------------------------------------------------------------------------------

PROCESS terrain_map_processing;

     Determine bin and altitude threshold using current X,Y coordinates;
     SVECT.TERALT = terrain altitude threshold;

     IF (approach zone bin)
          THEN SET AZBIN;
          ELSE CLEAR AZBIN;

     IF (SVECT.CONC EQ $TRUE OR SVECT.PSTAT EQ $TRUE)
          THEN IF (ZMCC LT SVECT.TERALT)
                    THEN SET TVIOL;
                    ELSE PERFORM adjacent_bin_checks;
               IF (TVIOL EQ $FALSE)
                    THEN Determine projection end point by projecting aircraft
                            straight ahead for TRHTM seconds;
                         LOOP;
                              Project aircraft to nearest X-crossing point,
                                 Y-crossing point, or end point;
                              IF (SVECT.ZD LT 0 AND
                                  new aircraft altitude LT old threshold)
                                   THEN SET TVIOL;
                                   ELSE PERFORM adjacent_bin_checks;
                         EXITIF (TVIOL EQ $TRUE OR end point reached);
                              Determine new bin and new threshold;
                              IF (SVECT.ZD GE 0 AND
                                  aircraft altitude LT new threshold)
                                   THEN SET TVIOL;
                         EXITIF (TVIOL EQ $TRUE);
                         ENDLOOP;
          ELSE :   < no terrain violation. >

END terrain_map_processing;
```

```
--------------------------------------------------------------------------------

ROUTINE ENTER_ALERT_IN_PWILST
   IN (Aircraft PWILST, alert type, and
       obstacle or restricted airspace identifier)
   OUT (PWILST entry);

     REPEAT WHILE (there are more entries in PWILST AND
                  no matching entry has been found);

          Select next entry in PWILST;
          IF (entry type EQ alert type)
             < including AIRSPACE_TYPE for TCA or restricted airspace alert >
               THEN IF (alert type EQ 'terrain alert' OR alert type EQ 'TCA alert')
                         THEN matching entry has been found;
                    ELSEIF (obstacle or restricted airspace identifier
                              matches identifier in PWILST entry)
                         THEN matching entry has been found;
                    OTHERWISE ;   < matching entry has not been found. >

     ENDREPEAT;

     IF (matching entry has been found)
          THEN Set END field = 0;
          ELSE Create new entry in PWILST below any TA entries;
               Set entry type = alert type;
                  < including AIRSPACE_TYPE for TCA or restricted airspace alert >
               Set END field = 0;
               Set FTAT field = 1;
               IF (alert type EQ 'obstacle alert' OR
                   alert type EQ 'restricted airspace alert')
                    THEN record obstacle or restricted airspace identifier
                              in PWILST entry;

END ENTER_ALERT_IN_PWILST;
```

```
ROUTINE ENTER_ALERT_IN_PWILST
   IN (Aircraft PWILST, alert type, ORAID)
   OUT (PWILST entry);

     BIT FOUND;

     CLEAR FOUND;
     REPEAT WHILE (there are more entries in PWILST AND FOUND EQ $FALSE);
          Select next entry in PWILST;
          IF (entry type EQ alert type)
               THEN IF (alert type EQ 'terrain alert' OR alert type EQ 'TCA alert')
                         THEN SET FOUND;
                    ELSEIF (alert type EQ 'obstacle alert AND
                              ORAID EQ OBSTACLE.OBSTACLE_NO)
                         THEN SET FOUND;
                    ELSEIF (alert type EQ 'restricted airspace alert' AND
                              ORAID EQ AIRSPACE.IDENTIFIER)
                         THEN SET FOUND;
                    OTHERWISE :   < matching entry has not been found. >
     ENDREPEAT;

     IF (FOUND EQ $TRUE)
          THEN CLEAR END in PWILST entry;
          ELSE Create new entry in PWILST below any TA entries;
               Set entry type = alert type;
               CLEAR END;
               SET FTAT;
               IF (alert type EQ 'obstacle alert')
                    THEN OBSTACLE.OBSTACLE_NO = ORAID;
               ELSEIF (alert type EQ 'restricted airspace alert')
                    THEN AIRSPACE.IDENTIFIER = ORAID;
               OTHERWISE :   < take no further action. >

END ENTER_ALERT_IN_PWILST;
```

```
--------------------------------------------------------------------------------

ROUTINE SEND_ALERT_TO_CONTROLLER
   IN (Aircraft ID, alert type)
   OUT (Controller alert message);

     < Build controller alert message. >

     Put DABS ID of aircraft in ACID1 field;
     Indicate alert type in ANTYP field;

     Indicate in CS1 field that aircraft is controlled;
     IF (aircraft is ATARS-equipped)
          THEN Indicate in CS1 field that aircraft is ATARS-equipped;
               Indicate in V1 field that controller voice communication
                  is not necessary;
          ELSE Indicate in CS1 field that aircraft is not ATARS-equipped;
               Indicate in V1 field possible need for controller voice
                  communication;

     Send alert message to controller;

END SEND_ALERT_TO_CONTROLLER;
```

```
-----------------------------------------------------------------------------------

ROUTINE SEND_ALERT_TO_CONTROLLER
   IN (SVECT, alert type)
   OUT (Controller alert message);

     TYPE_CODE = $CAM;
     ACID1 = SVECT.CODE;

     IF (alert type EQ 'terrain alert')
          THEN AMTYP = $TAM;
     ELSEIF (alert type EQ 'obstacle alert')
          THEN AMTYP = $OAM;
     OTHERWISE AMTYP = $RAM;

     IF (SVECT.ATSEQ NE $UNEQ)
          THEN CS1 = $COAT;
               V1 = $NOVOICE;
          ELSE CS1 = $COUN;
               V1 = $VOICE;

     Send alert message to controller;

END SEND_ALERT_TO_CONTROLLER;
```

7. COARSE SCREEN PROCESSING

The Coarse Screen Task is an operation applied to a single aircraft on the X-list or the EX-list. The executive program points to the initial aircraft for a particular sector's list on the X-list or EX-list and coarse screening then steps through the list, processing all the aircraft on the sector list. The purpose of coarse screen filtering is to identify aircraft which may be in conflict with the subject aircraft. This is done by computing x, y, and z search limits for a sector subject aircraft and then testing all aircraft in the appropriate linked list, which fall within the x limit against the y and z search limits, and a z rate test. By segregating aircraft through the use of the X-list and the EX-list, it is possible to construct a coarse screening procedure that can provide ample warning times for aircraft with exceptionally high speeds (those on the EX-list) without requiring unnecessarily large search volumes for the majority of the aircraft which are on the X-list.

The ATARS program may use larger look-ahead times for pairs involving controlled aircraft than for pairs involving only uncontrolled aircraft. Since the greatest number of aircraft is expected to be uncontrolled, a significant savings in computational requirements is achieved by using a separate coarse screening procedure with larger search limits for controlled aircraft. For coarse screening of uncontrolled subject aircraft on the X-list, only other uncontrolled aircraft with x coordinates greater than the x coordinate of the subject aircraft are examined. This constitutes a one-way search, in which only uncontrolled/uncontrolled potential conflicts are investigated. For controlled subject aircraft, a two-way search of the X-list is used. When the search is made in the positive x direction, all controlled and uncontrolled aircraft are investigated, and when the search is made in the negative x direction, only the uncontrolled aircraft are investigated. This avoids a duplicate declaration of controlled/controlled conflicts.

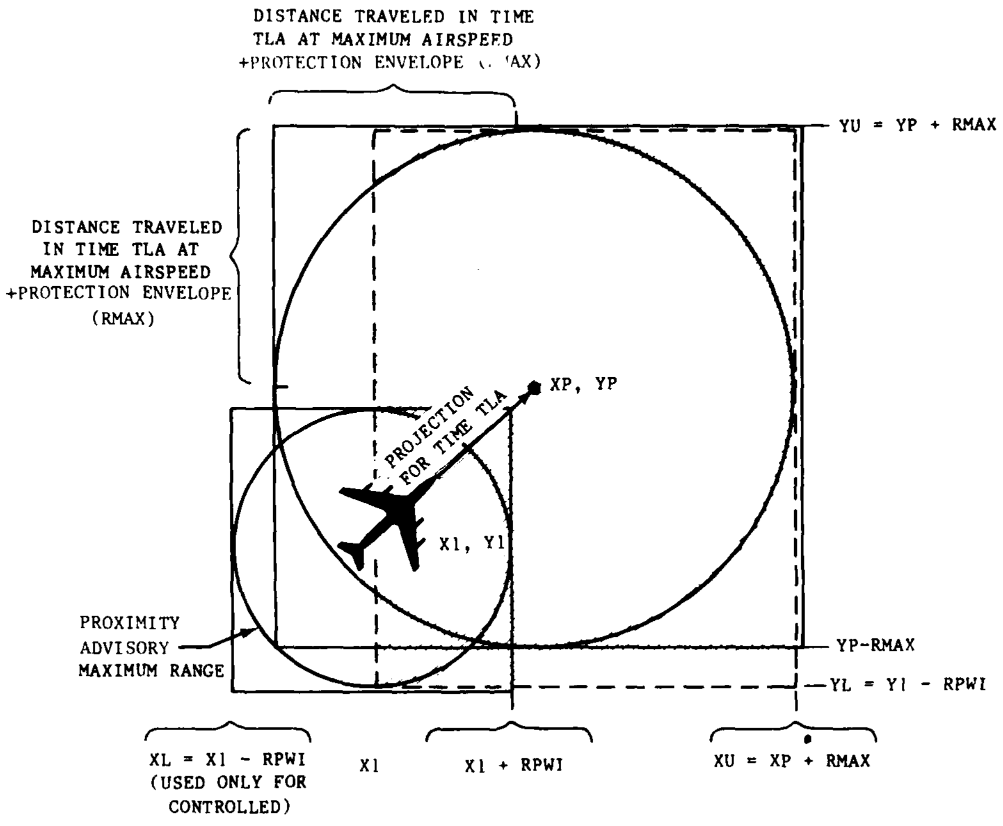
7.1 Coarse Screen Search Region

As shown in Figure 7-1, the search region implied by a given aircraft is computed as the outer bounds of either:

1. A region given by the coarse screen proximity advisory search criterion (this region is the same for both a controlled and an uncontrolled subject aircraft)

AD-A104 147 MITRE CORP MCLEAN VA METREK DIV F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-1 FAA-RD-81-45-1 NL

4 OF 5
AD
A104147


DISTANCE TRAVELED IN TIME TLA AT MAXIMUM AIRSPEED +PROTECTION ENVELOPE (RMAX)
YU = YP + RMAX
DISTANCE TRAVELED IN TIME TLA AT MAXIMUM AIRSPEED +PROTECTION ENVELOPE (RMAX)
XP, YP
PROJECTION FOR TIME TLA
X1, Y1
PROXIMITY ADVISORY MAXIMUM RANGE
YP-RMAX
YL = Y1 - RPWI
XL = X1 - RPWI (USED ONLY FOR CONTROLLED)
X1
X1 + RPWI
XU = XP + RMAX

WHERE	:	TLA	- TLV FOR UNCONTROLLED AND TLI FOR CONTROLLED
		TLV	- LONGEST LEAD TIME FOR UNCONTROLLED
		TLI	- LONGEST LEAD TIME FOR CONTROLLED
		RMAX	- RMAXV FOR UNCONTROLLED AND RMAXI FOR CONTROLLED
		RMAXV	- MAXIMUM RANGE OVER LONGEST LOOK AHEAD TIME FOR UNCONTROLLED AC PLUS PROTECTION ENVELOPE
		RMAXI	- MAXIMUM RANGE OVER LONGEST LOOK AHEAD TIME FOR CONTROLLED AC PLUS PROTECTION ENVELOPE
		RPWI	- LARGEST SEPARATION BETWEEN AIRCRAFT FOR A PROXIMITY ADVISORY ASSUMING BOTH AIRCRAFT AT MAXIMUM SPEED

FIGURE 7-1
SEARCH REGION FOR COARSE SCREENING OF UNCONTROLLED AIRCRAFT

2. A region based on a time projection using the subject aircraft tracked velocities and the largest look-ahead time appropriate for the subject aircraft, and assuming the intruding aircraft to have maximum velocity (240 knots low altitude, 600 knots high altitude)

The search limits so obtained will then permit detection of proximity advisories or any of the other messages which are based on the tau criterion. For the nominal values of parameters given in Figure 7-1, a 240 knot maximum speed has been assumed.

7.2 XUPFL Flag Update

The coarse screen resets the XUPFL flag for the subject aircraft which is used in the Aircraft Update Processing Task to prevent multiple updates during repositioning of aircraft on the X-list and EX-list. It is convenient to do this in coarse screening when each aircraft in the sector list is being accessed for coarse screening, rather than requiring a separate pass through the X-list and EX-list specifically to reset this flag.

7.3 Coarse Screen Checking

Coarse screening involves searching along a linked list from the position of the aircraft undergoing coarse screening to the upper and lower (if appropriate) X-limit. X-list aircraft with altitude reports are checked against other X-list aircraft only. EX-list aircraft with altitude reports are checked against EX-list aircraft and, if their altitude and vertical rate warrant, against X-list aircraft.

If the subject aircraft has no altitude report both X-list and EX-list must be searched for possible conflicts. Such an aircraft on the X-list is thus used for possible conflict searches on the EX-list (subject X-list/object EX-list search). When the subject non-mode C aircraft is on the EX-list, a search is also made of the X-list. This search excludes all X-list aircraft which are also non-mode C, since they have already been matched with EX-list aircraft in the subject X-list/object EX-list search.

For all aircraft encountered in these searches, the y test is applied and if both subject and object aircraft have an altitude report the z limit test and z rate limit test are applied. If either aircraft has no altitude, only the y limit test is used.

If these tests show a possible conflict, then a pair of aircraft which requires further processing has been identified.

Any individual aircraft is included on only the X-list or EX-list. Hence, when an aircraft on the EX-list must be tested against aircraft on the X-list, special provisions must be made for finding the place on the X-list to begin the search. A procedure analogous to that used for initial entry on the X-list is used in which the first signpost below the aircraft's X position is obtained and the X-list entered at that point. It is not important to locate the subject aircraft's exact position on the X-list; it is sufficient to obtain an entry point between the upper and lower limits. If the distance between signposts is small enough, this will happen automatically. Even if an entry point to the X-list were used, which fell outside the X-limits, the procedure would work correctly, but would be inefficient since more aircraft than necessary would be tested in coarse screening.

7.4 Potential Pair List

Each subject aircraft and object aircraft found in applying the coarse screen constitute a potential pair. The pairs are entered on the Potential Pair List. The Detect Task references each entry on this list and determines the exact type of hazard for the potential conflict, if any.

7.5 Pseudocode for Coarse Screen Task

The pseudocode for the Coarse Screen Task follows. This task operates only after the New Aircraft Processing Task and the Aircraft Update Processing Task have completed.

PSEUDOCODE TABLE OF CONTENTS

```
--------------------------------------------------------------------------------

STRUCTURE CSVBL
  GROUP limits
    FLT XL      <lower x boundary for X/EX-list search>
    FLT XU      <upper x boundary for X/EX-list search>
    FLT YL      <lower y boundary for X/EX-list search>
    FLT YU      <upper y boundary for X/EX-list search>
    FLT ZL      <lower z boundary for X/EX-list search>
    FLT ZU      <upper z boundary for X/EX-list search>
  GROUP predictions
    FLT XP      <x predicted position of subject AC>
    FLT YP      <y predicted position of subject AC>
    FLT ZP      <z predicted position of subject AC>
  GROUP bounds
    FLT TLA     <desired warning for proximity advisory in secs >
    FLT RMAX    <desired warning for proximity advisory in nmi>
  GROUP xclud_types
    BIT NO_CONT <if true, controlled AC are excluded from search>
    BIT NO_NONC <if true, non-mode C AC are excluded from search>
  GROUP starting_loc
    PTR START   <starting point of search in X/EX-list>
ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------

TASK COARSE_SCREEN

     <Searches X and EX-lists for pairs of AC that are sufficiently
      close or will be sufficiently close to require DETECT processing>

   IN  (X/EX-list for sector with non-mode C AC reports)
   OUT (potential conflict pairs);

     PERFORM x_list_search;
     PERFORM ex_list_search;

END COARSE_SCREEN;
```

---------------------- COARSE SCREEN TASK HIGH-LEVEL LOGIC ----------------------

```
--------------------------------------------------------------------------------

TASK COARSE_SCREEN
   IN  (SECPTE, SECPTX, CSCREEN, SVECT)
   OUT (potential conflict pairs);

     PTR SECPTE:    <location of first AC on X-list for this sector (from SIDSPX)>
     PTR SECPTX:    <location of first AC on EX-list for this sector (from SIDSPE)>

     PERFORM x_list_search;
     PERFORM ex_list_search;

END COARSE_SCREEN;
```

```
--------------------------------------------------------------------------------
PROCESS x_list_search;

    <Examines X-list for object AC that may possibly require an advisory
     message involving subject AC>

    REPEAT WHILE (more AC in X-list for this sector);

        CLEAR list updated flag in state vector;
        IF (AC controlled)
            THEN PERFORM compute_controlled_AC_search_limits;
                Determine starting point of forward search on X-list;
                CALL SEARCH_FORWARD;  <for all types of AC>
                Determine starting point of backward search on X-list;
                CALL SEARCH_BACKWARD; <for uncontrolled AC only>
            ELSE PERFORM compute_uncontrolled_AC_search_limits;
                Determine starting point of forward search on X-list;
                CALL SEARCH_FORWARD;  <for uncontrolled AC only>
        IF (AC is non-mode C)
            THEN PERFORM compute_high_altitude_search_limits;
                Determine starting point of forward search on EX-list;
                CALL SEARCH_FORWARD; <for all types of AC>
                Determine starting point of backward search on EX-list;
                CALL SEARCH_BACKWARD; <for all types of AC>

    ENDREPEAT;

END x_list_search;
```

```
--------------------------------------------------------------------------------

PROCESS x_list_search
     PTR NUSUBJ; <temporary local pointer to next subject AC (represented by SVECT1)>
     <N.B. in all calls to SEARCH_FORWARD/BACKWARD the following,
      IN  (CSVBL, SVECT1, CSCREEN)
      OUT (potential conflict pairs):  does not appear due to lack of space>
     NUSUBJ = SECPTX;
     REPEAT WHILE (SVECT1.NEXTX NE $NULL);
          CLEAR SVECT1.XUPL;
          IF (SVECT1.CONC EQ $TRUE)  <subject AC is controlled>
               THEN PERFORM compute_controlled_AC_search_limits;
                    CSVBL.START = SVECT1.NEXTX;
                    CSVBL.NO_CONT = $FALSE;
                    CSVBL.NO_NONC = $FALSE;
                    CALL SEARCH_FORWARD;  <for all types of AC>
                    CSVBL.START = SVECT1.PREVX;
                    CSVBL.NO_CONT = $TRUE;
                    CALL SEARCH_BACKWARD; <for uncontrolled AC only>
               ELSE PERFORM compute_uncontrolled_AC_search_limits;
                    CSVBL.START = SVECT1.NEXTX;
                    CSVBL.NO_CONT = $TRUE;
                    CSVBL.NO_NONC = $FALSE;
                    CALL SEARCH_FORWARD;  <for uncontrolled AC only>
          IF (SVECT1.HCFLG EQ $FALSE)
               THEN PERFORM compute_high_altitude_search_limits;
                    CSVBL.START = NEXTX variable as designated in ex signpost
                                       INT(SVECT1.X/CSCREEN.XSP);
                    CSVBL.NO_CONT = $FALSE;
                    CSVBL.NO_NONC = $FALSE;
                    CALL SEARCH_FORWARD; <for all types of AC>
                    CSVBL.START = PREVX variable as designated in ex signpost
                                       INT(SVECT1.X/CSCREEN.XSP);
                    CALL SEARCH_BACKWARD; <for all types of AC>
          NUSUBJ = SVECT1.NEXTX;
     ENDREPEAT;
END x_list_search;
----------------------  COARSE SCREEN TASK LOW-LEVEL LOGIC  ----------------------
```

```
--------------------------------------------------------------------------------

PROCESS ex_list_search

     <Examines EX-list for object AC that may possibly require an
      advisory message involving subject AC>

     REPEAT WHILE (more AC in EX-list for this sector)

          CLEAR list updated flag in state vector;
          PERFORM compute_high_altitude_search_limits;
          Determine starting point of forward search on EX-list;
          CALL SEARCH_FORWARD; <for all types of AC>
          IF (altitude threshold exceded or expected to be exceded
                in 2 minutes OR AC is non-mode C)
               THEN PERFORM compute_low_altitude_search_limits;
                    Determine starting point of forward search on X-list;
                    CALL SEARCH_FORWARD; <for all types of AC except non-mode C>
                    Determine starting point of backward search on X-list;
                    CALL SEARCH_BACKWARD; <for all types of AC except non-mode C>

     ENDREPEAT;

END ex_list_search;
```

```
--------------------------------------------------------------------------------

PROCESS ex_list_search;

     PTR NUSUBJ;   <temporary local pointer to next subject AC,
                    SVECT1 represents AC pointed to by NUSUBJ>

     NUSUBJ = SECPTE;
     REPEAT WHILE (SVECT.NEXTX NE $NULL);

          CLEAR SVECT1.XUPFL;
          PERFORM compute_high_altitude_search_limits;
          CSVBL.START = SVECT1.NEXTX;
          CSVBL.NO_CONT = $FALSE;
          CSVBL.NO_NONC = $FALSE;
          CALL SEARCH_FORWARD  <for all types of AC>
               IN  (CSVBL, SVECT1, CSCREEN)
               OUT (potential conflict pair);
          IF ((SVECT1.Z LT CSCREEN.AHI) OR (CSVBL.ZP LT AHI)
                    OR (SVECT1.MCFLG EQ $FALSE))
               THEN PERFORM compute_low_altitude_search_limits;
                    CSVBL.START = NEXTX variable as designated in x signpost
                                   INT(SVECT1.X/CSCREEN.XSP);
                    CSVBL.NO_NONC = $TRUE;
                    CALL SEARCH_FORWARD  <for all types of AC except non-mode C>
                         IN  (CSVBL, SVECT1, CSCREEN)
                         OUT (potential conflict pairs);
                    CSVBL.START = PREVX variable as designated in x signpost
                                   INT(SVECT1.X/CSCEEN.XSP);
                    CALL SEARCH_BACKWARD  <for all types of AC except non-mode C>
                         IN  (CSVBL, SVECT1, CSCREEN)
                         OUT (potential conflict pairs);
          NUSUBJ = SVECT1.NEXTX;
     ENDREPEAT;

END ex_list_search;

----------------------- COARSE SCREEN TASK LOW-LEVEL LOGIC -----------------------
```

```
--------------------------------------------------------------------------------

PROCESS compute_controlled_AC_search_limits;

     Define largest look ahead time for controlled AC; <TLI>
     Define maximum distance traveled by controlled AC; <RMAXI>
     CALL CALCULATE_SEARCH_LIMITS;

END compute_controlled_AC_search_limits;
```

```
--------------------------------------------------------------------------------

PROCESS compute_controlled_AC_search_limits;

     CSVBL.ELA = CSCREEN.ELI;
     CSVBL.RMAX = CSCREEN.RMAXI;
     CALL CALCULATE_SEARCH_LIMITS
          IN  (GROUP bounds, SVECT1, CSCREEN)
          OUT (GROUP limits, GROUP predictions);

END compute_controlled_AC_search_limits;
```

```
--------------------------------------------------------------------------------
PROCESS compute_high_altitude_search_limits;

     Define largest look ahead time for controlled AC; <TLI>
     Define maximum distance traveled by intruding AC in EX-list; <PMAX9>
     CALL CALCULATE_SEARCH_LIMITS;

END compute_high_altitude_search_limits;
```

```
-----------------------------------------------------------------------------------

PROCESS compute_high_altitude_search_limits;

     CSVBL.TLA = CSCREEN.TLI;
     CSVBL.RMAX = CSCREEN.RMAXH;
     CALL CALCULATE_SEARCH_LIMITS
          IN  (GROUP bounds, SVECT1, CSCREEN)
          OUT (GROUP limits, GROUP predictions);

END compute_high_altitude_search_limits;
```

```
--------------------------------------------------------------------------------
PROCESS compute_low_altitude_search_limits;

    Define maximum distance traveled by intruding AC in EX-list; <RMAXI>
    Calculate x & y upper and lower bounds of search;

END compute_low_altitude_search_limits;
```

```
--------------------------------------------------------------------------------
PROCESS compute_low_altitude_search_limits;

     FLT (XPU, XPL, YPU, YPL, ZPU, ZPL
          XRU, XRL, YRU, YRL, ZRU, ZRL); <temporary (local) variables>

     CSVBL.bounds.RMAX = CSCREEN.RMAXI;

     <TLA has been defined in a previous call to compute_high_altitude_search_limits>
     CSVBL.predictions.XP = SVECT1.X + SVECT1.XD * CSVBL.bounds.TLA;
     CSVBL.predictions.YP = SVECT1.y + SVECT1.YD * CSVBL.bounds.TLA;

     XPU = CSVBL.predictions.XP + CSVBL.bounds.RMAX;
     XPL = CSVBL.predictions.XP - CSVBL.bounds.RMAX;
     YPU = CSVBL.predictions.YP + CSVBL.bounds.RMAX;
     YPL = CSVBL.predictions.YP - CSVBL.bounds.RMAX;
     XRU = SVECT1.X + CSCREEN.RPWI;
     XRL = SVECT1.X - CSCREEN.RPWI;
     YRU = SVECT1.Y + CSCREEN.RPWI;
     YRL = SVECT1.Y - CSCREEN.RPWI;
     CSVBL.limits.XU = MAX(XPU,XRU);
     CSVBL.limits.XL = MIN(XPL,XRL);
     CSVBL.limits.YU = MAX(YPU,YRU);
     CSVBL.limits.YL = MIN(YPL,YRL);

END compute_low_altitude_search_limits;
```

```
--------------------------------------------------------------------------------

PROCESS compute_uncontrolled_AC_search_limits;

     Define largest look ahead time for uncontrolled AC; <TLV>
     Define maximum distance traveled by uncontrolled AC; <RMAXV>
     CALL CALCULATE_SEARCH_LIMITS;

END compute_uncontrolled_AC_search_limits;
```

```
--------------------------------------------------------------------------------

PROCESS compute_uncontrolled_AC_search_limits;

     CSVBL.TLA = CSCREEN.TLV;
     CSVBL.RMAX = CSCREEN.RMAXV;
     CALL CALCULATE_SEARCH_LIMITS
          IN  (GROUP bounds, SVECT1, CSCREEN)
          OUT (GROUP limits, GROUP predictions);

END compute_uncontrolled_AC_search_limits;
```

```
--------------------------------------------------------------------------------

ROUTINE SEARCH_BACKWARD
   IN  (starting address of search, search limits, table of
          possible object AC to be excluded in search -
          controlled or non-mode C)
   OUT (potential conflict pairs);

    <Given the lower bounds of a region, searches the X or EX-list
     for AC in the region>

    REPEAT WHILE (object AC x position above lower search bound area
                  OR list is not empty):

         IF (object AC is a 'signpost' OR is not in ATARS service area)
              OR is in table of excluded AC types)
              THEN; <do nothing>
         ELSEIF (object AC y position outside limits)
              THEN; <do nothing>
         ELSEIF (object AC non-mode C)
              THEN add AC to coarse screen potential pair list for this sector;
         ELSEIF (object altitude within search limits)
              THEN add AC to potential pair list for this sector;
         ELSEIF (object vertical velocity above threshold AND
                   AC pair will be co-altitude soon)
              THEN add AC to potential pair list for this sector;
         Get next preceding entry on list;

    ENDREPEAT;

END SEARCH_BACKWARD;
```

```
--------------------------------------------------------------------------------
ROUTINE SEARCH_BACKWARD
   IN  (CSVBL, SVECT1, CSCREEN)
   OUT (potential conflict pairs);

     PTR OBJAC; <temporary (local) pointer to object AC,
                SVECT2 represents object AC, SVECT1 represents subject AC>
     FLT ZTIME; <temporary (local) variable >

     OBJAC = CSVBL.START;
     REPEAT WHILE ((OBJAC NE $NULL)  OR (SVECT2.X GT CSVBL.XL));

          IF ((SVECT2.SPIDFG EQ $TRUE) OR (SVECT2.ATSS EQ $FALSE) OR
             (SVECT2.NCFLG EQ CSVBL.NO_NONC) OR
             (SVECT2.CONC EQ CSVBL.NO_CONT))
               THEN: <do nothing>
          ELSEIF (SVECT2.Y GT CSVBL.YU OR SVECT2.Y LT CSVBL.YL)
               THEN: <do nothing>
          ELSEIF (SVECT2.NCFLG EQ $FALSE)
               THEN add AC to potential pair list for this sector;
          ELSEIF ((SVECT2.Z LT CSVBL.ZU) AND (SVECT2.Z GT CSVBL.ZL))
               THEN add AC to potential pair list for this sector;
          ELSEIF ((SVECT2.ZD GT CSCREEN.ZFAST)
               THEN ZTIME = (SVECT1.Z-SVECT2.Z)/(SVECT2.ZD-SVECT1.ZD):
                    IF ((0.0 LT ZTIME) AND (ZTIME LT CSVBL.TLA))
                         THEN add AC to potential pair list for this sector;
          OBJAC = SVECT2.PREVX;

     ENDREPEAT;

END SEARCH_BACKWARD;
```

```
------------------------------------------------------------------------------------
ROUTINE CALCULATE_SEARCH_LIMITS
   IN  (prediction time and maximum area suitable for
           proximity advisory)
   OUT (search limits to check for intruding AC);

     <Calculates the region used in an X/EX-list search>

     Calculate horizontal (x,y) boundaries using the proximity
      advisory region search area;
     Calculate horizontal boundaries using predicted position
      based on subject AC velocities and look ahead time;
     Choose the outer bounds of the combined areas;

     IF(AC had altitude information)
          THEN calculate vertical (z) boundaries using proximity
                advisory region search area;
               Calculate vertical boundaries using predicted positions
                based on subject AC velocity and look ahead time;
               Choose the outer bounds of the combined areas;

END CALCULATE_SEARCH_LIMITS;
```

```
--------------------------------------------------------------------------------

ROUTINE CALCULATE_SEARCH_LIMITS
   IN  (GROUP bounds, SVECT1, CSCREEN)
   OUT (GROUP limits, GROUP predictions);

     FLT (XPU, XPL, YPU, YPL, ZPU, ZPL
          XRU, XRL, YRU, YRL, ZRU, ZRL);  <temporary (local) variables>

     CSVBL.predictions.XP = SVECT1.X + SVECT1.XD * CSVBL.bounds.TLA;
     CSVBL.predictions.YP = SVECT1.Y + SVECT1.YD * CSVBL.bounds.TLA;
     XPU = CSVBL.predictions.XP + CSVBL.bounds.RMAX;
     XPL = CSVBL.predictions.XP - CSVBL.bounds.RMAX;
     YPU = CSVBL.predictions.YP + CSVBL.bounds.RMAX;
     YPL = CSVBL.predictions.YP - CSVBL.bounds.RMAX;
     XRU = SVECT1.X + CSCREEN.RPWI;
     XRL = SVECT1.X - CSCREEN.RPWI;
     YRU = SVECT1.Y + CSCREEN.RPWI;
     YRL = SVECT1.Y - CSCREEN.RPWI;
     CSVBL.limits.XU = MAX(XPU,XRU);
     CSVBL.limits.XL = MIN(XPL,XRL);
     CSVBL.limits.YU = MAX(YPU,YRU);
     CSVBL.limits.YL = MIN(YPL,YRL);

     IF (SVECT1.MCFLG NE $FALSE)  <AC is mode C>
          THEN CSVBL.predictions.ZP = SVECT1.Z + SVECT1.ZD * CSVBL.bounds.TLA;
               ZPU = CSVBL.predictions.ZP + CSCREEN.VPCS;
               ZPL = CSVBL.predictions.ZP - CSCREEN.VPCS;
               ZRU = SVECT1.Z + CSCREEN.VPCS;
               ZRL = SVECT1.Z - CSCREEN.VPCS;
               CSVBL.limits.ZU = MAX(ZPU,ZRU);
               CSVBL.limits.ZL = MIN(ZPL,ZRL);
          ELSE; <do nothing>

END CALCULATE_SEARCH_LIMITS;
```

```
ROUTINE SEARCH_FORWARD
   IN  (starting address of search, search limits, table of
        possible object AC to be excluded in search -
         controlled or non-mode C)
   OUT (potential conflict pairs);

     <Given the upper bounds of a region, searches the X or EX-list
      for AC in the region>

     REPEAT WHILE (object AC x position below upper search bound area
                  OR list is not empty);

          IF (object AC is a 'signpost' OR is not in ATARS service area)
               OR is in table of excluded AC types)
               THEN; <do nothing>
          ELSEIF (object AC y position outside limits)
               THEN; <do nothing>
          ELSEIF (object AC non-mode C)
               THEN add AC to course screen potential pair list for this sector;
          ELSEIF (object altitude within search limits)
               THEN add AC to potential pair list for this sector;
          ELSEIF (object vertical velocity above threshold AND
                    ac pair will be co-altitude soon)
               THEN add AC to potential pair list for this sector;
          Get next succeeding entry on list;

     ENDREPEAT;

END SEARCH_FORWARD;
```

```
------------------------------------------------------------------------------

ROUTINE SEARCH_FORWARD
   IN  (CSVBL, SVECT1, CSCREEN)
   OUT (potential conflict pairs);

     PTR OBJAC; <temporary (local) variable>

     <N.B. SVECT2 represents object AC pointed to by OBJAC,
           SVECT1 represents subject AC>

     OBJAC = CSVBL.START;
     REPEAT WHILE ((OBJAC NE $NULL) OR (SVECT2.X LT CSVBL.XU));

          IF ((SVECT2.SPIDFG EQ $TRUE) OR (SVECT2.ATSS EQ $FALSE) OR
               (VECT2.NCFLG EQ CSVBL.NO_NONC) OR
               (SVECT2.CONC EQ CSVBL.NO_CONT))
               THEN; <do nothing>
          ELSEIF (SVECT2.Y GT CSVBL.YU OR SVECT2.Y LT CSVBL.YL)
               THEN; <do nothing>
          ELSEIF (SVECT2.NCFLG EQ $FALSE)
               THEN add AC to course screen potential pair list for this sector;
          ELSEIF ((SVECT2.Z LT CSVBL.ZU) AND (SVECT2.Z GT CSVBL.ZL))
               THEN add AC to potential pair list for this sector;
          ELSEIF ((SVECT2.ZD GT CSCREEN.ZFAST)
               THEN ZTIME = (SVECT1.Z-SVECT2.Z)/(SVECT2.ZD-SVECT1.ZD);
                    IF ((0.0 LT ZTIME) AND (ZTIME LT CSVBL.TLA))
                         THEN add AC to potential pair list for this sector;
          OBJAC = SVECT2.NEXTX;

     ENDREPEAT;

END SEARCH_FORWARD;
```

8. DETECT TASK

The function of the Detect Task is to examine the potential pairs generated by the Coarse Screen Task, and for each determine the need and type of subsequent ATARS processing. The output of the Detect Task is an entry on the Encounter List (ELENTRY). For each pair put on the list, all succeeding tasks check its entry for possible work to be performed. Specifically the Detect Task determines if ATARS has control of a BCAS/ATCRBS conflict, if a controller alert is required, if a proximity warning is required, if a threat advisory is required, if a resolution advisory is required, or if resolution deletion is required. It does not determine the type of resolution advisory or generate messages. This is left to future tasks.

8.1 BCAS/ATCRBS Control

Whenever a BCAS-equipped aircraft is within the ATARS service area, it is desired that ATARS resolve all conflicts that it sees. The BCAS must not be permitted to resolve the same conflicts, as it might select a different resolution. However, the BCAS is permitted to operate near the limits of ATARS service, so that it may provide advisories against threats outside the service area.

When the BCAS encounters a DABS-equipped threat, the ATARS site ID bits serve to permit or inhibit BCAS action, as defined in Reference 11. When the threat is non-beacon equipped or non-mode C ATCRBS, BCAS does not track it. However, when the threat is mode C ATCRBS, ATARS must inform the BCAS that the particular aircraft pair is under ATARS control. The ATCRBS Track Block Message inhibits the BCAS for this threat; its absence implies that ATARS does not see the threat.

The method employed models the BCAS aircraft's detection logic, with slightly less restrictive thresholds (Table 8-1). If BCAS is expected to give a threat or resolution advisory soon, a message is sent to BCAS using a track block of position data to identify the ATCRBS aircraft (Section 16.1.1.2). BCAS is thus informed to yield control of the potential conflict. This message is not sent for all BCAS/ATCRBS pairs within the ATARS area, but only for those whose BCAS would generate advisories. This reduces the communications load on the DABS data link and avoids unnecessary processing by BCAS.

TABLE 8-1

BCAS SENSITIVITY LEVEL DEPENDENT VARIABLES,
STRUCTURE BCSVBL

VARIABLE	VALUE AT GIVEN SENSITIVITY LEVEL					
	2	3	4	5	6	7
H1	NA	.00278	.00278	.00278	.004	.005 $\mathrm{nmi}^2/\mathrm{s}$
DMOD	NA	.1	.3	.5	1.3	1.6 nmi
TRTHR	NA	23	25	30	35	40 s
TVTHR	NA	23	25	30	35	40 s
RTHRTA	.35	.50	.75	1.5	2.0	2.5 nmi
H1TA	.002	.00278	.00278	.00278	.004	.005 $\mathrm{nmi}^2/\mathrm{s}$
DMODTA	.13	.2	.4	.6	1.5	1.9 nmi
TRTHTA	25	35	40	45	50	53 s
TVTHRTA	25	35	40	45	50	53 s

8.2 Proximity Advisories

Proximity advisories are issued to an aircraft pair whenever they are estimated to be within 30 (TLPSQ) seconds of each other or within a fixed range (RPMIN) and altitude (RST). It is possible to issue proximity advisories when one aircraft is non-mode C. In this case the co-altitude condition is assumed to always be satisfied. Additionally a horizontal tau test is performed for these aircraft. If satisfied, a proximity advisory is issued. The need for this additional possibility of alarm occurs when the aircraft pair are co-altitude and in a head-on geometry. The tau test is designed to give an earlier warning for this case than a range test, since the threat advisory test is not performed. When both altitudes are known this test is not necessary as the normal threat logic handles the situation. For non-mode C aircraft the proximity advisory is the only function performed by the Detect Task.

The PWIFLG flag, when set, signifies a proximity advisory is necessary for the pair.

8.3 Threat Advisories

A threat advisory is an indication that a dangerous situation is imminent and if the aircraft maintain present course a resolution advisory will be sent. The threat advisory conditions are: the aircraft not be in a final approach zone (see Section 8.7), the aircraft be proximate in space or be estimated to be proximate soon (tau tests). Only those aircraft pairs which are not diverging or within spatial constraints are subject to the threat advisory tests.

The FPWFLG flag is set for uncontrolled aircraft and the FPIFLG flag is set for controlled aircraft to indicate threat advisory messages are required.

8.4 Resolution Advisories

Aircraft predicted to be within ATARS separation criteria violation and not in a final approach zone are issued resolution advisories. Aircraft pairs are subjected to the resolution advisory tests if the pair has satisfied all threat advisory conditions. The aircraft pair then must be proximate in space and/or estimated, by use of the tau tests, to be near separation violation. If these conditions are satisfied the CMDFLG flag is set and if one of the aircraft is controlled the IFRFLG flag is set. The resolution advisory is generated in the Master Resolution Task when two of the latest three scans has had the

CMDFLG flag set (see Section 12.3.1). It is possible to give an immediate command, i.e., the resolution advisory is generated and sent the same scan. Basically the separation violation must be imminent, existing, or the aircraft form a dangerous flight geometry (Maneuvering Threat Logic Process) where one aircraft can turn into the other without sufficient time to predict the airspace violation and safely escape. In these cases the MTTFLG flag is set to bypass the two out of three window.

When the conditions to set the CMDFLG and IFRFLG are met, the setting of these flags may still be inhibited. The aircraft pair are not given advisories when, at the time of predicted closest approach in the horizontal direction, their predicted separation in the vertical dimension exceeds a threshold. This occurs when the aircraft are presently vertically proximate but diverging with a significant rate.

8.5 Controller Alerts

Controller alert initiation logic resembles the threat and resolution advisory determination function. Its purpose is entirely distinct, however. If either aircraft is controlled, it is desired to inform the controller of a possible conflict situation before a resolution advisory is necessary.

The CAFLG flag is set if either aircraft is in an area type requiring controller notification and the tau or proximity tests are satisfied, or the aircraft are in a dangerous geometry (parallel, offset and turning towards one another). If both aircraft are in a final approach zone 2, the proximity tests, only, are applied in order to further desensitize the logic to converging traffic patterns (see Section 8.7).

The controller alert message is generated (see Section 11) as soon as three of the latest five scans (a system variable) have had the CAFLG flag set. This filter can be overridden and an immediate message sent if the dangerous maneuver is detected or the aircraft separation violation is existing. An immediate alarm is designated by setting the ICAFLG. A controller alert is no longer given when the last three scans have had no controller alert flag set in Detect Task.

8.6 Parameter Selection

The majority of the various thresholds that appear in the Detect Task and its routines depend on a number of criteria for their determination. Those parameters (or thresholds) that are not true constants are in general assigned in the Tau and Proximity Threshold Determination Routine (See Section 8.8). First, the non-constant thresholds may depend on the control status of the aircraft in an encounter: controlled/controlled, controlled/uncontrolled, uncontrolled/uncontrolled. Additional specification may depend on area type of the encounter (1, 2, 3, 4), multiplicity of the encounter, and ATARS equipage. In the case of uncontrolled/uncontrolled encounters, specification may also rely on a computed index, UUIND, which is set in the Uncon/Uncon Index Determination Routine. Furthermore, certain tau thresholds are computed based on closing speed, and ultimately rely on the thresholds TCONV and TCONH determined in the Tau Proximity and Threshold Determination Routine. Such variables have a nominal value calculation, plus a maximum value and a minimum value. The nominal value must be restricted by the smallest value consistent with system safety and by the largest value feasible with an acceptable unwanted alarm rate. A synopsis of all detection variables may be found in Table 8-2, 8-3, 8-4, and 8-5. True constants are defined in Appendix A.

8.7 Area Type and Zone Determination

As indicated previously, selection of thresholds may depend on the area type of an encounter, ENAT, determined in the Encounter Area Type Determination Routine. This index is a combination of the individual area types, ACAT, determined in the Area Type Determination Routine. For each new position, i.e. every scan, of the aircraft in the pair under consideration, the aircraft's area type is determined by referencing a map (the area type data base described in Table 8-6) and from these two area types for the pair, the encounter area type is calculated. It is possible for the area types for an aircraft to be previously determined by the domino logic for this scan before entry into the Detect Task. The Area Type Determination Routine will be called from the Encounter Area Type Determination Routine only as necessary.

Each ACAT area type 1 or 2 defines a horizontal parallelogram, type 1 encompassing the immediate vicinity of an airfield, and type 2 encompassing the approach areas for each runway between specified altitudes. Area type 1 may, however, be further modified with "legs", or straight line segments that may be used to remove corners of the parallelogram. Type 3 is the balance of the airspace out to a range of RDIST (system parameter), beyond which is the area type 4. Both aircraft having an ACAT value of 3 would result in ENAT being 3. Also, both aircraft in different area type 2 regions would result in ENAT being 3 again. The complete mapping algorithm is specified in the

TABLE 8-2

CONTROLLER ALERT VARIABLES, STRUCTURE CAVBL

VARIABLE	VALUE			
	ENAT 1	ENAT 2	ENAT 3	ENAT 4
AFCON	NA	275 ft	375 ft	375 ft
MDCON2	NA	.5625 nmi^2	1.44 nmi^2	1.44 nmi^2
RCON2	NA	.5625 nmi^2	1.44 nmi^2	1.44 nmi^2
TCONH	Calculated in Routine TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION			
TCONV	Calculated in Routine TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION			

TABLE 8-3

PARAMETER DETERMINATION VARIABLES, STRUCTURE PDVBL

VARIABLE	VALUE			
	ENAT 1	ENAT 2	ENAT 3	ENAT 4
ACONTH	275 ft	275 ft	375 ft	375 ft
RCONTH	.75 nmi	.75 nmi	1.2 nmi	1.2 nmi
TWARN	36.8 s	36.8 s	36.8 s	44.8 s

TABLE 8-4

RESOLUTION ADVISORY VARIABLES, STRUCTURE RAVBL

VARIABLE	INDICES		VALUE
	CONTROL STATE (PRCONT)	ENAT	
AIFR	C/C[1]	1	750 ft
		2	750 ft
		3	750 ft
		4	750 ft
	C/U	1	750 ft
		2	750 ft
		3	750 ft
		4	750 ft
RIFR2	C/C	1	.5625 nmi^2
		2	.5625 nmi^2
		3	.5625 nmi^2
		4	.5625 nmi^2
	C/U	1	.5625 nmi^2
		2	.5625 nmi^2
		3	.5625 nmi^2
		4	.5625 nmi^2
AF	U/U	1	750 ft
		2	750 ft
		3	750 ft
		4	750 ft
	C/U	1	750 ft
		2	750 ft
		3	750 ft
		4	750 ft
RCMD2	U/U	1	1.0 nmi^2
		2	1.0 nmi^2
		3	1.0 nmi^2
		4	1.0 nmi^2
	C/U	1	.5625 nmi^2
		2	.5625 nmi^2
		3	1.0 nmi^2
		4	1.0 nmi^2

TABLE 8-4
(Continued)

VARIABLE	INDICES				VALUE[3]		
	CONTROL STATE (PRCONT)	ENAT	MULT	EQUIP (PREQ)	NOMINAL	MIN	MAX
TIFRH	C/C	1	GT3	E/E[2]	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	30	30 s
				E/U	TCONH-35	30	30 s
		2	GT3	E/E	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	30	30 s
				E/U	TCONH-35	30	30 s
		3	GT3	E/E	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	30	30 s
				E/U	TCONH-35	30	30 s
		4	GT3	E/E	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	38	38 s
				E/U	TCONH-35	38	38 s
	C/U	1	GT3	E/E	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	30	30 s
				E/U	TCONH-35	30	30 s
		2	GT3	E/E	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	30	30 s
				E/U	TCONH-35	30	30 s
		3	GT3	E/E	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	30	30 s
				E/U	TCONH-35	30	30 s
		4	GT3	E/E	TCONH-35	60	60 s
				E/U	TCONH-35	60	60 s
			LE3	E/E	TCONH-35	38	38 s
				E/U	TCONH-35	38	38 s
TIFRV	Same as TIFRH, except TCONH is replaced by TCONV in the NOMINAL column.						

TABLE 8-4
(Concluded)

VARIABLE	INDICES				VALUE[3]		
	CONTROL STATE (PRCONT)	ENAT	MULT	EQUIP (PREQ)	NOMINAL	MIN	MAX
TCMDH	C/U	1	GT3	E/E	TCONH-15	60	60 s
				U/E	TCONH-15	60	60 s
			LE3	E/E	TCONH-15	30	45 s
				U/E	TCONH-15	30	45 s
		2	GT3	E/E	TCONH-15	60	60 s
				U/E	TCONH-15	60	60 s
			LE3	E/E	TCONH-15	30	45 s
				U/E	TCONH-15	30	45 s
		3	GT3	E/E	TCONH-15	60	60 s
				U/E	TCONH-15	60	60 s
			LE3	E/E	TCONH-15	30	45 s
				U/E	TCONH-15	30	45 s
		4	GT3	E/E	TCONH-15	60	60 s
				U/E	TCONH-15	60	60 s
			LE3	E/E	TCONH-15	38	53 s
				U/E	TCONH-15	38	53 s
			UUIND[4]		VALUE		
	U/U	1	1		30 s		
			2		38 s		
		2	1		30 s		
			2		38 s		
		3	1		30 s		
			2		38 s		
		4	1		38 s		
			2		38 s		
TCMDV	Same as TCMDH, except TCONH is replaced by TCONV in the NOMINAL column						

[1]C = Controlled, U = Uncontrolled, referenced as PRCONT in pseudocode.
[2]E = ATARS Equipped, U = Unequipped, referenced as PREQ in pseudocode.
[3]The values of TCONH, TCONV are calculated in Routine TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION.
[4]UUIND is defined in Routine UNCON/UNCON_INDEX_DETERMINATION.

TABLE 8-5

THREAT ADVISORY VARIABLES, STRUCTURE TAVBL

VARIABLE	INDICES		VALUE
	CONTROL STATE (PRCONT)	ENAT	
AFIFR	C/C[1]	1	1000 ft
		2	1000 ft
		3	1000 ft
		4	1000 ft
	C/U	1	1000 ft
		2	1000 ft
		3	1000 ft
		4	1000 ft
MDFPI2	C/C	1	.5625 nmi^2
		2	.5625 nmi^2
		3	.5625 nmi^2
		4	.5625 nmi^2
	C/U	1	.5625 nmi^2
		2	.5625 nmi^2
		3	1.44 nmi^2
		4	1.44 nmi^2
RFIFR2	C/C	1	.5625 nmi^2
		2	.5625 nmi^2
		3	.5625 nmi^2
		4	.5625 nmi^2
	C/U	1	.5625 nmi^2
		2	.5625 nmi^2
		3	1.44 nmi^2
		4	1.44 nmi^2

TABLE 8-5
(Continued)

VARIABLE	INDICES: CONTROL STATE (PRCONT)	INDICES: ENAT	VALUE
AFPWI	U/U	1	1000 ft
		2	1000 ft
		3	1000 ft
		4	1000 ft
	C/U	1	1000 ft
		2	1000 ft
		3	1000 ft
		4	1000 ft
MDFPW2	U/U	1	1.0 nmi^2
		2	1.0 nmi^2
		3	1.0 nmi^2
		4	1.0 nmi^2
	C/U	1	.5625 nmi^2
		2	.5625 nmi^2
		3	1.44 nmi^2
		4	1.44 nmi^2
RFPWI2	U/U	1	1.0 nmi^2
		2	1.0 nmi^2
		3	1.0 nmi^2
		4	1.0 nmi^2
	C/U	1	.5625 nmi^2
		2	.5625 nmi^2
		3	1.44 nmi^2
		4	1.44 nmi^2

TABLE 8-5
(Continued)

VARIABLE	INDICES				VALUE[3]		
	CONTROL STATE (PRCONT)	ENAT	MULT	EQUIP	NOMINAL	MIN	MAX
TFIFRH	C/C[1]	1	GT3	E/E[2]	TCONH	60	68 s
				E/U	TCONH	60	68 s
			LE3	E/E	TCONH	30	60 s
				E/U	TCONH	30	60 s
		2	GT3	E/E	TCONH	60	68 s
				E/U	TCONH	60	68 s
			LE3	E/E	TCONH	30	60 s
				E/U	TCONH	30	60 s
		3	GT3	E/E	TCONH	60	68 s
				E/U	TCONH	60	68 s
			LE3	E/E	TCONH	30	60 s
				E/U	TCONH	30	60 s
		4	GT3	E/E	TCONH	60	68 s
				E/U	TCONH	60	68 s
			LE3	E/E	TCONH	38	68 s
				E/U	TCONH	38	68 s
	C/U	1	GT3	E/E	TCONH	60	68 s
				E/U	TCONH	60	68 s
			LE3	E/E	TCONH	30	60 s
				E/U	TCONH	30	60 s
		2	GT3	E/E	TCONH	60	68 s
				E/U	TCONH	60	68 s
			LE3	E/E	TCONH	30	60 s
				E/U	TCONH	30	60 s
		3	GT3	E/E	TCONH	60	68 s
				E/U	TCONH	60	68 s
			LE3	E/E	TCONH	30	60 s
				E/U	TCONH	30	60 s
		4	GT3	E/E	TCONH	60	68 s
				E/U	TCONH	60	68 s
			LE3	E/E	TCONH	38	68 s
				E/U	TCONH	38	68 s
TFIFRV	Same as TFIFRH, except TCONH is replaced by TCONV in the NOMINAL column.						

TABLE 8-5
(Concluded)

VARIABLE	INDICES				VALUE[3]		
	CONTROL STATE (PRCONT)	ENAT	MULT	EQUIP	NOMINAL	MIN	MAX
TFPWIH	C/U	1	GT3	E/E	TCONH	60	68 s
				U/E	TCONH	60	68 s
			LE3	E/E	TCONH	30	60 s
				U/E	TCONH	30	60 s
		2	GT3	E/E	TCONH	60	68 s
				U/E	TCONH	60	68 s
			LE3	E/E	TCONH	30	60 s
				U/E	TCONH	30	60 s
		3	GT3	E/E	TCONH	60	68 s
				U/E	TCONH	60	68 s
			LE3	E/E	TCONH	30	60 s
				U/E	TCONH	30	60 s
		4	GT3	E/E	TCONH	60	68 s
				U/E	TCONH	60	68 s
			LE3	E/E	TCONH	38	68 s
				U/E	TCONH	38	68 s

		UUIND[4]	VALUE
U/U	1	1	45 s
		2	53 s
	2	1	45 s
		2	53 s
	3	1	45 s
		2	53 s
	4	1	53 s
		2	53 s

TFPWIV Same as TFPWIH, except TCONH is replaced by TCONV in the NOMINAL column.

---

[1]C = Controlled, U = Uncontrolled, referenced as PRCONT in pseudocode.
[2]E = ATARS Equipped, U = Unequipped, referenced as PREQ in pseudocode.
[3]The values of TCONH, TCONV are calculated in Routine TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION.
[4]UUIND is defined in Routine UNCON/UNCON_INDEX_DETERMINATION.

TABLE 8-6

AREA TYPE DATA BASE

An ATARS area type data base for a particular ATARS site consists of type 1 areas, their associated type 2 areas, a type 3 area and a type 4 area.

RDIST = 83.3 nmi (for sensor azimuth jitter of .06 degrees) defines the maximum extent of type 3 area. Beyond this range is type 4.

Encompassing all type 1 areas and type 2 areas is the parallelogram defined by ZHMNX, ZHMXX, ZHMNY, ZHMXY which serves as a coarse filter in area determination routines. Locations outside this area are not subjected to type 1 or 2 determination.

The number of type 1 areas is given by NOI. For each type 1 area (up to 10 type 1 areas are allowed) the following information is required:

a) IDI - the name given to the type 1 area

b) A1, B1, C1, C2, A2, B2, C3, C4 - define the parallelogram which delimits the type 1 area

c) ZMIN, ZMAX - define the limits of the type 1 area in altitude

d) CARQ1 - determines if a controller alert is required for this type 1 area

e) NLEGS - the number of "legs" which delimit the type 1 area (allow up to 25 per type 1 area). The use of the legs allows the parallelogram defined in (b) to be modified into a convex polygon.

f) D1, E1, F1 - define one leg for each area type

TABLE 8-6
(Concluded)

An aircraft is in a type 1 area if its position (X, Y, Z) satisfies:

1) C1 LE (A1 * X + B1 * Y) LE C2

2) C3 LE (A2 * X + B2 * Y) LE C4

3) ZMIN LE Z LE ZMAX

4) (D1 * X + E1) LE F1 for each leg

The number of type 2 areas is given by NOII. For each type 2 area (allow up to 100 type 2 areas) the following information is required:

a) IDI - the name of the type 1 area with which this type 2 area is associated.

b) U1, V1, W1, W2, U2, V2, W3, W4 - define the parallelogram which delimits the type 2 area

c) HMIN, HMAX - define the altitude extent of the type 2 area

d) CARQ2 - determines if a controller alert is required for this type 2 area

An aircraft is in a type 2 area if its position (X, Y, Z) satisfies

1) W1 LE (U1 * X + V1 * Y) LE W2

2) W3 LE (U2 * X + V2 * Y) LE W4

3) HMIN LE Z LE HMAX

pseudocode (Section 8.8). A value of 4 for ENAT represents the most sensitive area, ENAT 1 being the least sensitive.

The final approach zone status of arriving aircraft, FAZ, contained in the State Vector, is utilized in the Detect Task, the Terrain/Airspace/Obstacle Avoidance Task as well as in the Master Resolution Task. Basically, the final approach zone is divided into two types, type 1 encompassing the airfield (and generally to a lower altitude than for area type 1), and type 2 encompassing a sloping rectangular region containing the normal approach path for each runway.

The parameter FAZ can have the following values upon entry into the Detect Task:

FAZ = -1,	not initialized for this aircraft, must be set by Detect Task
FAZ = 0,	aircraft is not in a final approach zone
FAZ = 1,	aircraft is in a final approach zone type 1
FAZ = 2,	aircraft is in a final approach zone type 2

The Final Approach Zone Determination Routine, called when FAZ is not initialized, is the same routine used in the Terrain/Airspace/Obstacle Avoidance Task. If FAZ has the value 1 or 2 then the State Vector parameter ZPRT has been initialized to the call letters of the airport associated with the final approach zone. Table 8-7 describes the zone data base.

The area types define the area of desensitization for the ATARS detection function. The zones define the region for inhibiting the resolution advisory function. Additionally, the zone 2 region also defines where controller alerts generated by prediction (tau tests) are to be inhibited. The proximity (immediate range) tests are never inhibited. Appropriate definition of this region will prevent nuisance alerts in parallel approach zones and converging approach zones. For this inhibit function to be applicable, both aircraft must be in a zone 2 region (not necessarily the same) associated with the same airport. Zone 2 should always be encompassed by area type 1 and/or 2.

TABLE 8-7

ZONE TYPE DATA BASE

An ATARS zone data base for a particular ATARS site consists of all the zone 1's and their associated zone 2's.

Encompassing all the zones, be they 1 or 2, for a site is the parallelogram defined by ZJMNX, ZJMXX, ZJMNY, ZJMXY, which serves as a coarse filter in zone determination. Locations outside this area are not subjected to zone 1 or 2 determination.

NOZ1 is the number of zone 1's for this site (a maximum of 10). For each zone 1 the following information is required:

a) IDZI - the name given to this zone 1

b) AZONL1, AZONW1, BZONL1, BZONW1, CZONL1, CZONW1, WZON1, LZON1 - define the parallelogram which delimits the zone 1

c) ZZON1 - defines the altitude extent of the zone 1

d) ACTZ1 - determines if this zone 1 is active or not

An aircraft is in a zone 1 if its position satisfies:

1) -WZON1 LE (AZONW1 * X + BZONW1 * Y + CZONW1) LE WZON1

2) -LZON1 LE (AZONL1 * X + BZONL1 * Y + CZONL1) LE LZON1

3) Z LE ZZON1

NOZ2 determines the number of zone 2's present in an ATARS site. A maximum of 100 zone 2's are allowed. For each zone 2 the following information is required:

a) IDZ1 - the name of the zone 1 area with which this zone 2 is associated

b) AZONL2, BZONL2 - the north and east components, respectively of a normal horizontal vector parallel to the main axis of the given zone 2 and pointing away from the airfield

c) WZON2, AZONW2, BZONW2, CZONW2, LZON2, CZONL2 - with (b) defines the parallelogram in the horizontal plane which delineates the zone 2

TABLE 8-7
(Concluded)

d) ZZON2, AZONZ2, BZONZ2, CZONZ2, DZONZ2, - define the linear surface in 3-space which defines the approach slope of the zone 2

e) COAA2 - defines the deviation from the normal defined by (b) within which an aircraft is considered to be in zone 2. COAA2 = .9698

f) ACTZ2 - determines if this zone 2 is active or not.

An aircraft is in a given zone 2 if its position (X, Y, Z) and horizontal velocity (XD, YD) satisfy:

1) -WZON2 <u>LE</u> (AZONW2 * X + BZONW2 * Y + CZONW2) <u>LE</u> WZON2

2) -LZON2 <u>LE</u> (AZONL2 * X + BZONL2 * Y + CZONL2) <u>LE</u> LZON2

3) -ZZON2 <u>LE</u>
(AZONZ2 * X + BZONZ2 * Y + CZONZ2 * Z + DZONZ2)
<u>LE</u> ZZON2

4) (XD * AZONL2 + YD * BZONL2) <u>LT</u> 0

5) (XD * AZONL2 + YD * BZONL2)^2 <u>GE</u> (XD^2 + YD^2) * COAA^2

Construction of the zones adheres to the following guidelines:

1. The zone 1 extends approximately .5 nmi on either side of a runway.

2. The zone 1 ends where the runway ends.

3. The zone 1 must be a parallelogram (There can be more than one zone 1 associated with a given airport, which could remove the restriction).

4. The zone 1 should be approximately 500 feet high.

5. The zone 2 extends approximately .5 nmi on either side of the centerline extending through the runway.

6. The zone 2 cannot overlap with another zone 2 (a software restriction).

7. The zone 2 begins at the outer marker and extends to the zone 1.

8. The zone 2 should be approximately 400 feet in depth.

8.8 Pseudocode for Detect Task

The pseudocode for the Detect Task follows. This task can operate as soon as the Coarse Screen Task has entered a pair on the Potential Pair List. Any qualified variable or parameter name (e.g., ELENTRY.TCONV) which does not appear in the list of local variables and parameters in the beginning of the low level pseudocode, belongs to a system data structure which is defined in Section 3.3. Similarly unqualified names (e.g., ADOT) are local to the Detect Task and appear in one of three structures, MISCVBL, PATHVBL, ELVBL. No distinction has been made between variables local to a low level process and used only within that process and variables local to the Detect Task and used by different processes within this task. An example of the latter would be MULT and of the former RXVS.

The routines Encounter Area Type Determination, Area Type Determination, Final Approach Zone Determination, are referenced by other ATARS tasks. These routines appear in the pseudocode as local to the Detect Task. Care must be taken in determining their exact form in any computer installation, as the code may need to be modified to allow for multiple entries from different tasks.

Frequent abbreviations used in the pseudocode are: RA for resolution advisory, TA for threat advisory, and CA for controller alert.

PSEUDOCODE TABLE OF CONTENTS

PSEUDOCODE TABLE OF CONTENTS

```
------------------------------------------------------------------------------

<*** MANEUVERING TARGET DETECTION PARAMETERS (APP A) ***>

STRUCTURE MTPARM
  GROUP cntr_thresholds      <controller alert thresholds>
     FLT CAMR2               <range threshold>
     FLT CAMA                <altitude separation threshold>
     FLT CAMVSQ              <velocity threshold (sq)>
     FLT CAMCP2              <cosine thresh (sq) for parallelism determination>
     FLT CAMRM2              <identical to MTTRM2, below>
     FLT CAMSB2              <sine thresh (sq) for offset/intrail determination>
  GROUP gnl_thresholds       <for general maneuvering threat detection>
     FLT MTTR2               <range thresh>
     FLT MTTA                <altitude separation threshold>
     FLT MTTVSQ              <velocity threshold (sq)>
     FLT COSP2               <cosine threshold (sq) for parallelism determination>
     FLT MTTRM2              <value to prevent division by zero>
     FLT MTTSB2              <sine threshold (sq) for offset/intrail determination>
ENDSTRUCTURE;
```

```
-------------------------- DETECT TASK LOCAL PARAMETERS --------------------------
```

```
------------------------------------------------------------------------------------

<*** PROXIMITY ADVISORY PARAMETERS (APP A) ***>

STRUCTURE PAPARM
  GROUP thresholds
     FLT RPMIN              <minimum proximity range (sq)>
     FLT TLPSQ              <proximity time parameter (sq)>
     FLT VP1                <proximity altitude threshold>
ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------------

<***   STRUCTURE NATAPARM  (APP A) ***>

STRUCTURE NATAPARM    <used in non-mode C threat detection>
  GROUP nathrs
     FLT R2NA    <range below which traffic advisory is always given>
     FLT THNA    <tau horizontal threshold for generating traffic advisory>
     FLT MD2NA  <predicted miss distance threshold for generating TA>
     FLT TSEPSQ <time estimate of minimum separation before traffic advisory is given>
ENDSTRUCTURE;
```

```
-----------------------------------------------------------------------------

<*** CONTROLLER ALERT PARAMETERS  (APP A) ***>


STRUCTURE CAPARM

  GROUP zone2

     FLT ZAPCON             <minimum vertical separation for AC in Zone 2>

     FLT ZRCON2             <minimum range (squared) threshold for AC in Zone 2>

ENDSTRUCTURE;
```

```
-------------------------------------------------------------------------------------

<*** CONTROLLER ALERT VARIABLES (Table 8-2) ***>

STRUCTURE CAVBL
  GROUP thresholds
     FLT AFCON            <immediate altitude threshold>
     FLT MDCON2           <miss distance threshold (squared)>
     FLT RCON2            <range threshold (squared)>
     FLT TCONH            <horizontal tau threshold>
     FLT TCONV            <vertical tau threshold>
ENDSTRUCTURE;
```

```
-------------------------- DETECT TASK LOCAL VARIABLES ----------------------------
```

```
--------------------------------------------------------------------------------

<*** RESOLUTION ADVISORY VARIABLES (Table 8-4) ***>


STRUCTURE RAVBL
  GROUP ctl_thresholds
     FLT AIFR               <immediate altitude threshold>
     FLT RIFR2              <range threshold (squared)>
     FLT TIFRH              <horizontal Tau threshold>
     FLT TIFRV              <vertical Tau threshold>
  GROUP unc_thresholds
     FLT AF                 <immediate altitude threshold>
     FLT RCMD2              <range threshold (squared)>
     FLT TCMDH              <horizontal Tau threshold>
     FLT TCMDV              <vertical Tau threshold>
ENDSTRUCTURE;



<*** THREAT ADVISORY VARIABLES (Table 8-5) ***>


STRUCTURE TAVBL
  GROUP ctl_thresholds
     FLT AFIFR              <immediate altitude threshold>
     FLT MDFPI2             <miss distance threshold (squared)>
     FLT RFIFR2             <range threshold (squared)>
     FLT TFIFRH             <horizontal Tau threshold>
     FLT TFIFRV             <vertical Tau threshold>
  GROUP unc_thresholds
     FLT AFPWI              <immediate altitude threshold>
     FLT MDFPW2             <miss distance threshold (squared)>
     FLT RFPWI2             <range threshold (squared)>
     FLT TFPWIH             <horizontal Tau threshold>
     FLT TFPWIV             <vertical Tau threshold>
ENDSTRUCTURE;



-------------------------- DETECT TASK LOCAL VARIABLES ---------------------------
```

```
---------------------------------------------------------------------------

<***   STRUCTURE BCSVBL   ***>

STRUCTURE BCSVBL  <see Table 8-1 & Appendix A for assigned values>
   GROUP res
     FLT RDTHR      <range rate threshold, limits tau in parallel flight tracks>
     FLT H1         <threshold for divergence range hit test for RA>
     FLT DMOD       <horizontal clearance used to define collision threshold>
     FLT TRTHR      <threshold of time to closest approach>
     FLT ZTHR       <threshold of altitude separation>
     FLT ZDTHR      <altitude rate divergence threshold>
     FLT TVTHR      <threshold of time to closest approach>
   GROUP threat
     FLT RTHRTA     <range  threshold  for TA>
     FLT RDTHRTA    <range rate threshold for TA>
     FLT H1TA       <threshold for divergence range hit test for TA>
     FLT DMODTA     <performs role of DMOD for TA>
     FLT TRTHRTA    <performs role of TRTHR for TA>
     FLT ZTHRTA     <performs role of ZTHR for TA>
     FLT ZDTHRTA    <performs role of ZDTHR for TA>
     FLT TVTHRTA    <performs role of TVTHR for TA>
ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------------

<*** ENCOUNTER CHARACTERISTIC VARIABLES ***>

STRUCTURE ELVBL
  GROUP local
     BIT INFAZ2;            <both AC in Zone 2 (along glide slope)>
     INT MULT               <multiplicity of encounter>
     INT PRCONT             <summarizes control state of ac pair>
     INT PREQ               <summarizes equp. state of ac pair>
     FLT VRAT               <ratio of eq/uneq AC speeds>
     INT UUIND              <unc/unc index>
     INT SSL                <BCAS sensitivity level for AC pair>
ENDSTRUCTURE;
```

```
-------------------------------------------------------------------------------------

<*** LOGIC-PATH VARIABLES ***>

STRUCTURE PATHVBL
  GROUP local
      BIT MT_DETECTED        <maneuvering target detected>
      BIT PF_FAILED          <prefiltering failed>
      BIT HPROX              <horizontal proximity detected>
      BIT VPROX              <vertical proximity detected>
      BIT BCSOFF             <pair under ATARS control only>
      BIT NOCA               <no controller alert needed>
      BIT GOTMT              <go to maneuvering target threat logic>
      BIT EXITLOOP           <jump out of loop>
      BIT DONEBOTH           <both aircraft in pair processed in MTT>
      BIT NORES              <no RA needed for this pair>
      BIT FILTFAIL           <inhibit RA due to vertical divergence>
      BIT NOTHREAT           <no TA needed for this pair>
ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------

<*** MISCELLANEOUS VARIABLES ***>

STRUCTURE MISCVBL
  GROUP local
     FLT RX      <x component of range between AC (nmi)>
     FLT RY      <y component of range between AC (nmi)>
     FLT RZ      <altitude separation of two AC (ft)>
     FLT R       <horizontal range between two AC (nmi)>
     FLT THTRU   <tau true, unmodified (no DSQ) horizontal tau (s)>
     FLT VRX     <closing x velocity (nmi/s)>
     FLT VRY     <closing y velocity (nmi/s)>
     FLT VRZ     <closing z velocity (ft/s)>
     FLT VRZA    <altitude converging (negative), diverging rate (ft/s)>
     FLT TLA     <maximium prediction time used in Detect (s)>
     FLT COSA2   <cosine of angle between AC velocity vectors>
     FLT TM      <vertical divergence prediction time used (s)>
     FLT TZ1     <predicted altitude of AC1 after TM seconds (ft)>
     FLT TZ2     <predicted altitude of AC2 after TM seconds (ft)>
     FLT TVMD    <predicted vertical separation of AC after TM seconds (ft)>
     FLT AH      <horizontal immediate command set prediction time threshold (s)>
     FLT AV      <vertical immediate command set prediction time threshold (s)>
     FLT VR2     <closing velocity squared, (nmi/s)**2>
     FLT RST     <range threshold to determine proximity advisory (nmi**2)>
     FLT RD      <closing range rate in BCAS inhibit logic (nmi/s)>
     FLT RDTA    <closing range rate in BCAS inhibit logic <nmi/s)>
     FLT DSQ     <range modification used in calculation of th, nmi/(s**2)>
     FLT A       <altitude separation of AC in BCAS inhibit logic (ft)>
     FLT ADOT    <altitude closing rate used in BCAS inhibit logic (ft/s)>
     FLT RXVS    <cross product of position vector connecting AC & a velocity vector>
     FLT SINB2   <sine of angle between a velocity vector and position vector>
     FLT RDTEMP <closing range rate in BCAS inhibit logic (nmi/s)>
     FLT TAUR    <modified time to minimum separation in BCAS inhibit logic (s)>
     FLT TRTRU   <unmodified time to minimum separation in BCAS inhibitlogic (s)>
ENDSTRUCTURE;

-------------------------  DETECT TASK LOCAL VARIABLES  --------------------------
```

```
--------------------------------------------------------------------------------
TASK DETECT
  IN (Potential conflict pair with state vectors and conflict tables)
  OUT (Encounter list entry);

   <This task determines if traffic, threat or resolution advisories are required>

     Reserve space for possible ELENTRY structure;
     IF (either AC is non-mode C);
          THEN PERFORM missing_altitude_traffic_advisory;
          ELSE PERFORM variable_initialization;
               IF (one AC BCAS equipped AND other AC ATCRBS equipped);
                    THEN PERFORM BCAS_inhibit_algorithm;
               PERFORM AC_converging_or_proximate_determination;  <prefiltering>
               IF (pair passed prefiltering);
                    THEN PERFORM number_of_additional_AC_in_conflict_determination;
                         CALL MINIMUM_APPROACH_DISTANCE_PREDICTION;
                         PERFORM parameter_selection;  <determine protection
                                   envelopes and time thresholds for alerts>
                         IF (at least one AC controlled);
                              THEN PERFORM controller_alert_determination;
                         IF (at least one AC eqdipped AND both AC not in final
                              approach zone);
                              THEN PERFORM threat_advisory_determination;
                                   PERFORM resolution_advisory_determination:
               IF (at least one AC equipped);
                    THEN PERFORM proximity_advisory_determination;
                         PERFORM maneuvering_threat_logic;
     PERFORM succeeding_processing_flag_determination;
     IF (any further ATARS processing required)
          THEN PERFORM uninitialized_variable_computation;
               Store ELENTRY in Encounter List;
          ELSE Release ELENTRY space;

END DETECT;

------------------------- DETECT TASK HIGH-LEVEL LOGIC --------------------------
```

```
--------------------------------------------------------------------------------

TASK DETECT
   IN (SVECT1, SVECT2, CSCREEN, conflict tables and pair recs (if any))
   OUT (encounter list entry with flags set);
     Allocate ELENTRY;
     IF (SVECT1.MCFLG EQ $FALSE OR SVECT2.MCFLG EQ $FALSE)
          THEN PERFORM missing_altitude_traffic_advisory;
          ELSE PERFORM variable_initialization;
               IF ((SVECT1.ATSEQ EQ $ABEQ OR SVECT2.ATSEQ EQ $ABEQ)
                     AND (SVECT1.TYPE EQ $ATCRBS OR SVECT2.TYPE EQ $ATCRBS))
                    THEN PERFORM BCAS_inhibit_algorithm;
               PERFORM AC_converging_or_proximate_determination;

               IF (PF_FAILED = $FALSE)
                    THEN PERFORM number_of_additional_AC_in_conflict_determination;
                         CALL MINIMUM_APPROACH_DISTANCE_PREDICTION
                              IN  (VRX,VRY)
                              OUT (ELENTRY.MD2);
                         PERFORM parameter_selection;
                         IF (PRCONT NE $NOCONT)
                              THEN PERFORM controller_alert_determination;

                         IF ((PREQ NE $NOEQ) AND
                               (SVECT1.FAZ = $FAZO AND SVECT2.FAZ = $FAZO))
                              THEN PERFORM threat_advisory_determination;
                                   PERFORM resolution_advisory_determination;

               IF (PREQ NE $NOEQ)
                    THEN PERFORM proximity_advisory_determination;
                         PERFORM maneuvering_threat_logic;
     PERFORM succeeding_processing_flag_determination;
     IF (any of GROUP ELENTRY.processing_required NE $FALSE)
          THEN PERFORM uninitialized_variable_computation;
               Link ELENTRY to Encounter List;
          ELSE Deallocate ELENTRY;
END DETECT;
-------------------------- DETECT TASK LOW-LEVEL LOGIC ---------------------------
```

```
PROCESS missing_altitude_traffic_advisory;

    <Given potential conflict pair with one or both AC lacking
     altitude information, this process determines if proximity advisory is needed.
     N.B. all thresholds are not dependent on area types or
     controller alert state>

    Clear encounter list entry flags;  <future processing and Detect flags>
    Calculate range between AC;
    IF (range LT minimum range)
        THEN declare proximity advisory is necessary;
        ELSE  compute separation assuming AC maneuver towards one
               another for 30 seconds (tlpsq);
            IF (current range LT separation approximation)
                THEN declare proximity advisory is necessary;
                ELSE calculate time to closest horizontal approach (th),
                      minimum horizontal miss distance (md2);
                    IF (th LT threshold AND md2 LT threshold)
                        THEN declare proximity advisory is necessary;

END missing_altitude_traffic_advisory;
```

```
-------------------------------------------------------------------------------------
PROCESS missing_altitude_traffic_advisory;

     <Given potential conflict pair with one or both AC lacking
      altitude information, determines if proximity advisory is needed.
      N.B.,  all thresholds are not dependent on area types or
      controller alert state>

     CLEAR GROUP ELENTRY.flags;
     CLEAR GROUP ELENTRY.processing_required;
     RX =  SVECT2.X - SVECT1.X;
     RY =  SVECT2.Y - SVECT1.Y;
     ELENTRY.RANGE2 =  RX*RX + RY*RY;
     IF (ELENTRY.RANGE2 LT NATAPARM.R2NA)
          THEN SET PWIFLG;
          ELSE RST = 2*NATAPARM.TSEPSQ*(SVECT1.VSQ +SVECT2.VSQ);
               IF (ELENTRY.RANGE2 LT RST);
                    THEN SET PWIFLG;
                    ELSE VRX = SVECT2.XD - SVECT1.XD;
                         VRY = SVECT2.YD - SVECT1.YD;
                         ELENTRY.DOT = (RX * VRX) + (RY * VRY);
                         DSQ = DETPARM.BDET+(DETPARM.ADET*(SVECT1.VSQ+SVECT2.VSQ));
                         ELENTRY.TH = -(ELENTRY.RANGE2 - DSQ) / ELENTRY.DOT;
                         CALL MINIMUM_APPROACH_DISTANCE_PREDICTION
                               IN   (VRX,VRY)
                               OUT  (ELENTRY.MD2);
                         IF ((ELENTRY.TH LT NATAPARM.THNA) AND
                                                  (ELENTRY.MD2 LT NATAPARM.MD2NA))
                              THEN SET PWIFLG;

END missing_altitude_traffic_advisory;
```

```
--------------------------------------------------------------------------------
PROCESS variable_initialization;

     <Performs preliminary convergence calculations and initializes variables>

     Clear encounter list entry;

     Indicate that (encounter area type,
                    miss distance,
                    horizontal Tau,
                    vertical Tau) are uninitialized;

     Indicate that resolution advisory UWC Tau thresholds are uninitialized;

     Compute (vertical separation,
              convergence rate,
              range);

     Determine equipage and control status;

END variable_initialization;
```

```
-------------------------------------------------------------------------------------

PROCESS variable_initialization;

     CLEAR GROUP ELENTRY.flags;
     CLEAR GROUP ELENTRY.processing_required;

     ELENTRY.EMAT = $UDAT;
     ELENTRY.MD2 = $UDMD;
     ELENTRY.TH = $UDTAU;
     ELENTRY.TV = $UDTAU;
     RAVBL.TCMDH = $UDTAU;
     RAVBL.TCMDV = $UDTAU;

     RZ = SVECT2.Z - SVECT1.Z;
     ELENTRY.ALT = ABS(RZ);
     RX = SVECT2.X - SVECT1.X;
     RY = SVECT2.Y - SVECT1.Y;
     VRX = SVECT2.XD - SVECT1.XD;
     VRY = SVECT2.YD - SVECT1.YD;
     ELENTRY.DOT = (RX * VRX) + (RY * VRY);
     ELENTRY.RANGE2 = RX**2 + RY**2;
     R = SQRT(RANGE2);

     CALL AIRCRAFT_PAIR_EQUIPMENT_AND_CONTROL_STATE_DETERMINATION
          IN  (SVECT1, SVECT2)
          OUT (PRCONT,PREQ);

END variable_initialization;
```

```
--------------------------------------------------------------------------------

PROCESS BCAS_inhibit_algorithm;

     <Determines if BCAS threat logic should be inhibited for this pair>

     IF (BCAS sensitivity level GE 2)   <is BCAS operating?>
          THEN IF (this site is primary for BCAS AC)
                    THEN PERFORM BCAS_threat_logic;
                    ELSE PERFORM BCAS_resolution_determination_logic;

END BCAS_inhibit_algorithm;
```

```
--------------------------------------------------------------------------------
PROCESS BCAS_inhibit_algorithm;

     <Determines if BCAS should be inhibited for this pair>

     IF (SVECT1.ATSEQ EQ $ABEQ)
          THEN SSL = SVECT1.BCASSL;    < SSL is used to reference Table 8-1>
               PRIM = SVECT1.PSTAT;
          ELSE SSL = SVECT2.BCASSL;
               PRIM = SVECT2.PSTAT;
     IF (SSL GE 2)
          THEN IF (PRIM EQ $TRUE)
                     THEN PERFORM BCAS_threat_logic;
                     ELSE PERFORM BCAS_resolution_determination_logic;
          ELSE; <do nothing as BCAS not operating>

END BCAS_inhibit_algorithm;
```

```
----------------------------------------------------------------------------------------
PROCESS AC_converging_or_proximate_determination:

     <Serves as a prefilter - if AC are diverging or not near one
      another then no CA, RA, TA tau testing is performed>

     IF (the aircraft are diverging)
          THEN prefiltering failed;
          ELSE modify slightly diverging AC to slowly converging, if necessary;
               PERFORM tau_calculation;

               IF (horizontal Tau GE coarse-screen lookahead)
                    THEN IF (current range GE immediate range threshold)
                              THEN prefiltering failed;

               IF (prefiltering hasn't failed yet)
                    THEN IF (vertical Tau GE coarse-screen lookahead)
                              THEN IF (current altitude separation GE immediate
                                        altitude threshold)
                                        THEN prefiltering failed;

END AC_converging_or_proximate_determination;
```

```
------------------------------------------------------------------------------------
PROCESS AC_converging_or_proximate_determination;

     CLEAR PF_FAILED;

     IF (ELENTRY.DOT GT DETPARM.DOTTH)
          THEN SET PF_FAILED;
          ELSE IF (ELENTRY.DOT LT -DETPARM.DOTTH)
                    THEN;
                    ELSE ELENTRY.DOT = -DETPARM.DOTTH;
               PERFORM tau_calculation;

               IF (PRCONT EQ $NOCONT)
                    THEN TLA = CSCREEN.TLI;
                    ELSE TLA = CSCREEN.TLV;
               IF (ELENTRY.TH GE TLA)
                    THEN IF (ELENTRY.RANGE2 GE DETPARM.RDET)
                              THEN SET PF_FAILED;

               IF (PF_FAILED NE $TRUE)
                    THEN IF (ELENTRY.TV GE TLA)
                              THEN IF (ELENTRY.ALT GE DETPARM.AFDET)
                                        THEN SET PF_FAILED;

END AC_converging_or_proximate_determination;
```

```
PROCESS number_of_additional_AC_in_conflict_determination:

    <Determines parameter MULT which is used as an index in look-up
     tables for DETECT thresholds >

    IF (neither AC is in a conflict table)
        THEN set multiplicity to 2;

    ELSEIF (only one AC is in a conflict table)
        THEN set multiplicity to number of AC in table (NAC) + 1;

    ELSEIF (both aircraft are in the same conflict table)
        THEN set multiplicity to NAC;

    OTHERWISE <aircraft in different tables>
            set multiplicity to NAC(AC1) + NAC(AC2);

END number_of_additional_AC_in_conflict_determination;
```

```
--------------------------------------------------------------------------------

PROCESS number_of_additional_AC_in_conflict_determination;

    IF (SVECT1.CTPTR = NULL AND SVECT2.CTPTR = NULL)
        THEN NULT = 2;

    ELSEIF (SVECT1.CTPTR = NULL OR SVECT2.CTPTR = NULL)
        THEN access conflict table;
            NULT = CTHEAD.NAC + 1;

    ELSEIF (SVECT1.CTPTR EQ SVECT2.CTPTR)
        THEN NULT = CTHEAD.NAC;

    OTHERWISE <aircraft in different tables>
            NULT = CTHEAD.NAC(AC1) + CTHEAD.NAC(AC2);

END number_of_additional_AC_in_conflict_determination;
```

```
--------------------------------------------------------------------------------
PROCESS parameter_selection;

    <Determines (1) the protection envelope,
                (2) the predicted minimum miss distance threshold
                (3) the time to minimum approach threshold for RA, TA, CA.
     In general an alert is given if the AC violate the protection
     envelope, ie. are proximate, or, the time to minimum approach
     is short and the closest approach is small, ie. pass tau checks>

    IF (at least one AC is controlled)
        THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION;
             CALL TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION;
<select parameters used to decide if RA, TA, CA needed>

    ELSEIF (at least one aircraft is equipped)
        THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION;
             IF (both AC are in a final approach zone)
                 THEN: <no RA or TA given in FAZ, so don't get these thresholds>
                 ELSE CALL TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION;

END parameter_selection;
```

```
-------------------------------------------------------------------------------------

PROCESS parameter_selection;

     IF (PRCONT NE $NOCONT)
          THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION
                    IN  (SVECT1, SVECT2)
                    OUT (ELENTRY.ENAT, INFAZ);
               CALL TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION  <set parameters>
                        IN    (ENAT, MULT, PREQ, PRCONT)
                        INOUT (VRZA);

     ELSEIF (PREQ NE $NOEQ)
          THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION
                    IN  (SVECT1, SVECT2)
                    OUT (ELENTRY.ENAT, INFAZ);
               IF ((SVECT1.FAZ NE $FAZ0) AND (SVECT2.FAZ NE $FAZ0))
                    THEN;      < don't set parameters >
                    ELSE CALL TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION
                              IN    (ENAT, MULT, PREQ, PRCONT)
                              INOUT (VRZA);

END parameter_selection;
```

```
--------------------------------------------------------------------------------
PROCESS controller_alert_determination;

     IF (neither AC requires a controller alert)
          THEN:     < forego contoller alert processing >
     ELSEIF (both AC in glide slope)      < Zone 2 >
          THEN IF ((range LT minimum range in Zone 2) AND
                   (vertical separation LT minimum separation in Zone 2))
                    THEN SET(CAFLG, ICAFLG);
                    ELSE no alert necessary;
     OTHERWISE IF (AC are close in horizontal dimension)
                    THEN indicate horizontal proximity;
               ELSEIF (AC will be at minimum separation soon)
                    THEN:     < CA still a possibility >
               OTHERWISE failed horizontal tests;

               IF (AC pair within vertical CA violation envelope)
                    THEN indicate vertical proximity;
               ELSEIF (AC will be coaltitude soon)
                    THEN:     < CA still possible >
               OTHERWISE failed vertical tests;

               IF (predicted minimum separation (MD2) outside
                     horizontal protection envelope)
                    THEN failed miss distance test;

               IF (all tests passed)
                    THEN SET CAFLG;
               IF (horizontal proximity AND vertical proximity)
                    THEN SET ICAFLG;     < need CA this scan >
               ELSEIF (either AC turning)
                    THEN CALL AC_PARALLEL_OFFSET_TURNING_DETERMINATION;
                         IF (AC parallel, offset and turning towards one another>
                              THEN SET(CAFLG, ICAFLG);
END controller_alert_determination;

------------------------- DETECT TASK HIGH-LEVEL LOGIC -------------------------
```

```
-------------------------------------------------------------------------------------
PROCESS controller_alert_determination;

     CLEAR (NOCA, HPROX, VPROX);

     IF (SVECT1.CAREQ = $FALSE AND SVECT2.CAREQ = $FALSE)
          THEN:     < forego controller alerts >
     ELSEIF (INFAZ2 = $TRUE)
          THEN IF ((ELENTRY.RANGE2 LT CAPARM.ZRCON2) AND
                   (ELENTRY.ALT LT CAPARM.ZAFCON))
                    THEN SET(ELENTRY.CAFLG, ELENTRY.ICAFLG);
                    ELSE:
     OTHERWISE IF (ELENTRY.RANGE2 LT CAVBL.RCON2)
                    THEN SET HPROX;
               ELSEIF (ELENTRY.TH LT CAVBL.TCONH)
                    THEN:     < CA still a possibility >
               OTHERWISE SET NOCA;
               IF (ELENTRY.ALT LT CAVBL.AFCON)
                    THEN SET VPROX;
               ELSEIF (ELENTRY.TV GE 0 AND ELENTRY.TV LT CAVBL.TCONV)
                    THEN:     < CA still possible >
               OTHERWISE SET NOCA;
               IF (ELENTRY.HD2 GT CAVBL.HDCON2)
                    THEN SET NOCA;

               IF (NOCA = $FALSE)
                    THEN SET ELENTRY.CAFLG;
               IF (HPROX EQ $TRUE AND VPROX EQ $TRUE)
                    THEN SET ELENTRY.ICAFLG;     < need CA right away >
               ELSEIF (SVECT1.TURN NE $STRAIGHT OR SVECT2.TURN NE $STRAIGHT)
                    THEN CALL AC_PARALLEL_OFFSET_TURNING_DETERMINATION
                                    IN  (GROUP HTPARM.cntr_thresholds)
                                    OUT (HT_DETECTED);
                         IF (HT_DETECTED = $TRUE)
                              THEN SET(ELENTRY.CAFLG, ELENTRY.ICAFLG);
END controller_alert_determination;
--------------------------  DETECT TASK LOW-LEVEL LOGIC  ---------------------------
```

```
--------------------------------------------------------------------------------
PROCESS threat_advisory_determination;

     <Sets FPWFLG and FPIFLG to indicate a threat message is required>

     IF (at least one AC is uncontrolled and equipped)
          THEN CALL THREAT_TAU_AND_PROXIMITY_COMPARISONS;
                    < routine to determine threat advisory need >

     IF (at least one AC is controlled and equipped)
          THEN CALL THREAT_TAU_AND_PROXIMITY_COMPARISONS;
                    < routine to determine threat advisory need >

     <If AC pair is uncontrolled/controlled then THREAT_TAU_AND
      _PROXIMITY_COMPARISONS is called twice (once for each AC)
      with different TA thresholds for each call. If both AC
      controlled or both AC uncontrolled THREAT_TAU_AND_PROXIMITY
      _COMPARISONS is called once>

END threat_advisory_determination;
```

```
------------------------------------------------------------------------------------

PROCESS threat_advisory_determination;

     IF ((SVECT1.CUNC EQ $FALSE AND SVECT1.ATSEQ NE UNEQ) OR
         (SVECT2.CUNC EQ $FALSE AND SVECT2.ATSEQ NE UNEQ))
         THEN CALL THREAT_TAU_AND_PROXIMITY_COMPARISONS
                   < routine to determine threat advisory need >
                      IN  (GROUP TAVBL.unc_thresholds)
                      OUT (ELENTRY.FPWFLG);

     IF ((SVECT1.CUNC EQ $TRUE AND SVECT1.ATSEQ NE UNEQ) OR
         (SVECT2.CUNC EQ $TRUE AND SVECT2.ATSEQ NE UNEQ))
         THEN CALL THREAT_TAU_AND_PROXIMITY_COMPARISONS
                   < routine to determine threat advisory need >
                      IN  (GROUP TAVBL.ctl_thresholds)
                      OUT (ELENTRY.FPIFLG);

END threat_advisory_determination;
```

```
PROCESS resolution_advisory_determination;

     <Sets CMDFLG to indicate RA required, and if RA needed by controlled
      AC, also, sets IFRFLG>

     IF (threat advisory required for uncontrolled AC)      < FPWFLG >
          THEN CALL RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS;
                    < routine to determine resolution advisory need, sets CMDFLG >

     IF (threat advisory required for controlled AC)      < FPIFLG >
          THEN CALL RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS;
                    < routine to determine resolution advisory need, sets IFRFLG >
               IF (both AC controlled)
                    THEN CMDFLG = IFRFLG;

     <If AC pair is uncontrolled/controlled then RESOLUTION_TAU_AND
      _PROXIMITY_COMPARISONS is called twice (once for each AC)
      with different RA thresholds for each call. If both AC
      controlled or both AC uncontrolled RESOLUTION_TAU_AND_PROXIMITY
      _COMPARISONS is called once>

END resolution_advisory_determination;
```

```
--------------------------------------------------------------------------------
PROCESS resolution_advisory_determination;

     IF (ELENTRY.FPWFLG = $TRUE)
          THEN CALL RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS
                    < routine to determine resolution advisory need >
                       IN  (GROUP RAVBL.unc_thresholds)
                       OUT (ELENTRY.CMDFLG);

     IF (ELENTRY.FPIFLG = $TRUE)
          THEN CALL RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS
                    < routine to determine resolution advisory need >
                       IN  (GROUP RAVBL.ctl_thresholds)
                       OUT (ELENTRY.IFRFLG);
               IF (both AC controlled)
                    THEN ELENTRY.CMDFLG = ELENTRY.IFRFLG;

END resolution_advisory_determination;
```

```
--------------------------------------------------------------------------------------------------

PROCESS proximity_advisory_determination;

     <Sets PWIFLG to indicate that a proximity message is needed>

     IF (vertical separation GE proximity threshold)
          THEN;     < proximity advisory not needed >

     ELSEIF (range LT minimum range)
          THEN SET PWIFLG;     < need proximity advisory >

     OTHERWISE compute separation after time TLPSQ;
               IF (range LT computed separation)
                    THEN SET PWIFLG;

END proximity_advisory_determination;
```

```
--------------------------------------------------------------------------------
PROCESS proximity_advisory_determination;

     IF (ELENTRY.ALT GE PAPARM.VP1)
          THEN;     < proximity advisory not needed >

     ELSEIF (ELENTRY.RANGE2 LT PAPARM.RPMIN)
          THEN SET ELENTRY.PWIFLG;     < definitely need one >

     OTHERWISE RST = 2 * PAPARM.TLPSQ * (SVECT1.VSQ + SVECT2.VSQ);
               IF (ELENTRY.RANGE2 LT RST)
                    THEN SET ELENTRY.PWIFLG;

END proximity_advisory_determination;
```

```
--------------------------------------------------------------------------------
PROCESS maneuvering_threat_logic;

     <Following tests weed out situations that do not involve
      a maneuvering target threat>

     IF (immediate alarm already determined)      < MTTFLG >
          THEN:      < no need to perform this logic >
     ELSEIF (neither AC is turning)
          THEN:      < no maneuvering AC involved >
     ELSEIF (AC diverging at a sufficiently fast rate)
          THEN:      < no threat maneuver exists >
     ELSEIF (proximity advisory not already given)   < PWIFLG >
          THEN:      < not close enough to be concerned about maneuvering AC>
     ELSEIF: (both AC have been determined to be in a final approach zone)
          THEN:      < do not test as RA not given in final approach zone >
     OTHERWISE CALL AC_PARALLEL_OFFSET_TURNING_DETERMINATION; <manuever target>

               IF (maneuvering target threat detected)
                    THEN SET MTTFLG;
                         IF (encounter area type is uninitialized)
                              THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION;
                         IF (both AC in final approach zones)
                              THEN CLEAR MTTFLG;
                         ELSEIF (at least one AC is controlled and equipped)
                              THEN SET (CMDFLG, FPIFLG, IPPFLG);
                         ELSEIF (at least one AC is uncontrolled and equipped)
                              THEN SET (CMDFLG, FPWFLG);

END maneuvering_threat_logic;
```

```
--------------------------------------------------------------------------

PROCESS maneuvering_threat_logic;

     IF (ELENTRY.MTTFLG = $TRUE)
          THEN;
     ELSEIF (SVECT1.TURN = $STRAIGHT AND SVECT2.TURN = $STRAIGHT)
          THEN;
     ELSEIF (ELENTRY.DOT GT DETPARM.DOTTH)
          THEN;
     ELSEIF (ELENTRY.PWIFLG = $FALSE)
          THEN;
     ELSEIF ((SVECT1.FAZ EQ $FAZ1 OR SVECT1.FAZ EQ $FAZ2) AND
             (SVECT2.FAZ EQ $FAZ1 OR SVECT2.FAZ EQ $FAZ2))
          THEN;
     OTHERWISE CALL AC_PARALLEL_OFFSET_TURNING_DETERMINATION
                     IN  (GROUP MTPARM.gnl_thresholds)
                     OUT (MT_DETECTED);

               IF (MT_DETECTED = $TRUE)
                    THEN SET ELENTRY.MTTFLG;
                         IF (ELENTRY.ENAT = $UDAT)
                              THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION;
                         IF ((SVECT1.FAZ NE $FAZ0) AND (SVECT2.FAZ NE $FAZ0))
                              THEN CLEAR ELENTRY.MTTFLG;
                         ELSEIF ((SVECT1.CUNC EQ $TRUE AND SVECT1.ATSEQ NE $UNEQ)
                             OR  (SVECT2.CUNC EQ $TRUE AND SVECT2.ATSEQ NE $UNEQ))
                              THEN SET (ELENTRY.CHDFLG, ELENTRY.FPIFLG,
                                                ELENTRY.IFRFLG);
                         ELSEIF ((SVECT1.CUNC EQ $FALSE AND SVECT1.ATSEQ NE $UNEQ)
                             OR  (SVECT2.CUNC EQ $FALSE AND SVECT2.ATSEQ NE $UNEQ))
                              THEN SET (ELENTRY.CHDFLG, ELENTRY.FPWFLG);

END maneuvering_threat_logic;
```

```
---------------------------------------------------------------------------------
PROCESS succeeding_processing_flag_determination;

     <Given Detect output flags and conflict table (if it
      exists), determines future processing for this pair>

     IF (proximity advisory OR threat advisory needed)
          THEN mark encounter entry for Traffic advisory processing;
     IF (resolution advisory declared)
          THEN mark encounter entry for Master Resolution processing;
     IF (controller alert declared)
          THEN mark encounter entry for controller alert preview processing;
     IF (conflict pair record exists and no resolution advisory declared)
          THEN mark encounter entry for resolution deletion processing;
     IF (BCAS inhibit has been declared)
          THEN mark encounter entry for BCAS inhibit;

END succeeding_processing_flag_determination;
```

```
PROCESS succeeding_processing_flag_determination;

     <Given Detect output flags and conflict table (if it
      exists), determines future processing for this pair>

     IF (PWIFLG EQ $TRUE OR FPIFLG EQ $TRUE OR FPWFLG EQ $TRUE)
          THEN SET ELENTRY.TAREQ;
     IF (CMDFLG EQ $TRUE)
          THEN SET ELENTRY.RAREQ);
     IF (CAFLG EQ $TRUE)
          THEN SET ELENTRY.CNAREQ);
     IF (conflict pair record exists AND CMDFLG EQ $FALSE)
          THEN SET ELENTRY.BDREQ);
     IF (BCSOFF EQ $TRUE)
          THEN SET ELENTRY.BOFFREQ);

END succeeding_processing_flag_determination;
```

```
--------------------------------------------------------------------------------
PROCESS uninitialized_variable_computation;

     <If advisory is needed and all variables which describe encounter
      for future tasks are not calculated, then determine their values>

     IF (horizontal Tau is uninitialized)
          THEN PERFORM tau_calculation;

     IF (resolution advisory uncontrolled vertical Tau threshold is uninitialized)
          THEN IF (encounter area type is uninitialized)
                    THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION;
                         PERFORM number_of_additional_AC_in_conflict_determination;
               CALL TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION;

     IF (miss distance is uninitialized)
          THEN CALL MINIMUM_APPROACH_DISTANCE_PREDICTION:

END uninitialized_variable_computation;
```

```
------------------------------------------------------------------------------------
PROCESS uninitialized_variable_computation;

     IF (ELENTRY.TH = $UDTAU)
          THEN PERFORM tau_calculation;

     IF (RAVBL.TCMDV = $UDTAU)
          THEN IF (ELENTRY.ENAT = $UDAT)
                    THEN CALL ENCOUNTER_AREA_TYPE_DETERMINATION
                              IN  (SVECT1,SVECT2)
                              OUT (ELENTRY.ENAT, INFAZ);
                         PERFORM number_of_additional_AC_in_conflict_determination;
               CALL TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION
                         IN    (ENAT, MULT, PREQ, PRCONT)
                         INOUT (VRZA);

     IF (ELENTRY.MD2 = $UDMD)
          THEN CALL MINIMUM_APPROACH_DISTANCE_PREDICTION
                    IN  (VRX,VRY)
                    OUT (ELENTRY.MD2);

END uninitialized_variable_computation;
```

```
------------------------------------------------------------------------------------------

PROCESS BCAS_resolution_determination_logic:

     <Determines if a BCAS resolution advisory is imminent for this pair>

     Calculate range rate; <rd>
     IF (AC converging or nearly converging)
          THEN IF (AC are nearly converging)
                    THEN redefine range rate to be converging;
               Calculate time estimates for closest approach; <taur,trtru>
               IF (taur less than threshold)
                    THEN PERFORM vertical_test_for_RA;
     ELSEIF (r*rd less than threshold AND range less than threshold)
          THEN define value for tau true; <trtru>
               PERFORM vertical_test_for_RA;

END BCAS_resolution_determination_logic;
```

```
PROCESS BCAS_resolution_determination_logic;

     <Determines if a BCAS resolution advisory is imminent for this pair>

     CLEAR BCSOFF;
     RDTEMP = ELENTRY.DOT / R;  <R is calculated in variable_initialization >
     RD = RDTEMP:
     IF (RD LE BCSVBL.res.RDTHR)
          THEN IF (RD GE -BCSVBL.res.RDTHR)
                    THEN RDTEMP = -BCSVBL.res.RDTHR;
               TAUR = -(R - BCSVBL.res.DMOD) / RDTEMP;
               TRTRU = - R / RDTEMP;
               IF (TAUR LT BCSVBL.res.TRTHR(SSL);
                    THEN PERFORM vertical_test_for_RA;
     ELSEIF (R * RD LT BCSVBL.res.H1(SSL) AND R LT BCSVBL.res.DMOD(SSL));
          THEN TRTRU = BCSVBL.res.TLARGE;
               PERFORM vertical_test_for_RA;

END BCAS_resolution_determination_logic;
```

```
PROCESS BCAS_threat_logic;

     <Determines if a BCAS threat advisory is imminent for this pair.
      This logic encompasses BCAS resolution thresholds.>

     Calculate range rate; <rd>
     IF (AC are close in range) <r>
          THEN PERFORM vertical_test_for_TA;
     ELSEIF (AC are closing or nearly converging)
               THEN IF (AC are nearly converging)
                         THEN redefine range rate to be converging;
                    Calculate time estimate to closest approach; <taurta>
                    IF (closest approach will occur soon)
                         THEN PERFORM vertical_test_for_TA;
     ELSEIF (r*rd LT threshold AND range LT threshold)
          THEN PERFORM vertical_test_for_TA;

END BCAS_threat_logic;
```

```
---------------------------------------------------------------------------------------
PROCESS BCAS_threat_logic;

     <Determines if a BCAS threat advisory is imminent for this pair.
      This logic encompasses BCAS resolution thresholds>

     CLEAR BCSOFF;

     RD = ELENTRY.DOT / R; <R is calculated in variable_initialization>
     RDTA = RD;
     IF (R LT BCSVBL.threat.RTHRTA(SSL))
          THEN PERFORM vertical_test_for_TA;
     ELSEIF (RD LT BCSVBL.threat.RDTHTTA)
               THEN IF (RD GT -BCSVBL.threat.RDTHRTA)
                         THEN RDTA = -RDTHRTA;
                    TAURTA = - (R - BCSVBL.threat.DHOTA) / RDTA;
                    IF (TAURTA LT BCSVBL.threat.TRTHRTA(SSL))
                         THEN PERFORM vertical_test_for_TA;
     ELSEIF (R * RD LT BCSVBL.threat.H1TA(SSL) AND R LT BCSVBL.threat.DHODTA(SSL))
          THEN PERFORM vertical_test_for_TA;

END BCAS_threat_logic;
```

```
--------------------------------------------------------------------------------

PROCESS tau_calculation;

    Calculate horizontal modified tau;
    Calculate horizontal true tau;
    Calculate vertical tau;

END tau_calculation;
```

```
--------------------------------------------------------------------------------

PROCESS tau_calculation;

     DSQ = DETPARM.BDET + (DETPARM.ADET * (SVECT1.VSQ + SVECT2.VSQ));
     ELENTRY.TH = -(ELENTRY.RANGE2 - DSQ) / ELENTRY.DOT;
     THTRU = RANGE2 / ELENTRY.DOT;

     VRZ = SVECT2.ZD - SVECT1.ZD;
     VRZA = VRZ * SIGN(RZ);

     IF (VRZA LE DETPARM.VRZTH)
          THEN IF (VRZA GT -DETPARM.VRZTH)
                    THEN VRZA = -DETPARM.VRZTH;

     ELENTRY.TV = -ELENTRY.ALT / VRZA;

END tau_calculation;
```

```
--------------------------------------------------------------------------------------
PROCESS vertical_test_for_RA;

     <Determines if RA for BCAS AC imminent, having found horizontal
      dimension imminent>

     Calculate altitude separation and rate; <a, adot>
     IF (AC are close in vertical)
          THEN set flag to indicate BCAS inhibit for the pair; <RA imminent>
     ELSEIF (AC are closing in vertical)
               THEN calculate estimate to coaltitude: <tauvra>
                    IF (coaltitude soon)
                         THEN set flag to indicate BCAS inhibit for this pair;

END vertical_test_for_RA;
```

```
--------------------------------------------------------------------------------
PROCESS vertical_test_for_RA;

     <Determines if RA for BCAS AC imminent, assuming horizontal
      permissive case, no vmd logic is used>

     A = ELENTRY.RZ;
     ADOT = SVECT2.ZD - SVECT1.ZD;
     IF (A LT BCSVBL.res.ZTHR)
          THEN SET BCSOFF;
     ELSEIF (ADOT LT BCSVBL.res.ZDTHR)
               THEN TAUV = - A / ADOT;
                    IF (TAUV LT BCSVBL.res.TVTHR(SSL))
                         THEN SET BCSOFF;

END vertical_test_for_RA;
```

```
--------------------------------------------------------------------------------------

PROCESS vertical_test_for_TA;

     <Determines if TA for BCAS AC imminent, having found horizontal
      dimension imminent>

     Calculate altitude separation and rate; <a, adot>
     IF (AC are close in vertical)
          THEN set flag to indicate BCAS inhibit for this pair;  <TA imminent>
     ELSEIF (AC are closing in vertical)
               THEN calculate estimate to coaltitude; <tauvta>
                    IF (coaltitude soon)
                         THEN set flag to indicate BCAS inhibit for this pair;

END vertical_test_for_TA;
```

```
--------------------------------------------------------------------------------

PROCESS vertical_test_for_TA;

     <Determines if TA for BCAS AC imminent, assuming horizontal
      permissive case>

     A = ELENTRY.RZ;
     ADOT = SVECT2.ZD - SVECT1.ZD;
     IF (A LT BCSVBL.threat.ZTHRTA)
          THEN SET BCSOFF;
     ELSEIF (ADOT LT BCSVBL.threat.ZDTHRTA)
               THEN TAUVTA = - A / ADOT;
                    IF (TAUVTA LT BCSVBL.threat.TVTHRTA(SSL))
                         THEN SET BCSOFF;

END vertical_test_for_TA;
```

AD-A104 147 MITRE CORP MCLEAN VA METREK DIV F/G 17/7
AUTOMATIC TRAFFIC ADVISORY AND RESOLUTION SERVICE (ATARS) ALGOR--ETC(U)
JUN 81 R H LENTZ, W D LOVE, T L SIGNORE DOT-FA80WA-4370
UNCLASSIFIED MTR-81W120-1 FAA-RD-81-45-1 NL

5 OF 5
AD
A104147

END
DATE
FILMED
10-81
DTIC

```
-----------------------------------------------------------------------------------
ROUTINE AC_PARALLEL_OFFSET_TURNING_DETERMINATION
   IN (thresholds)
   OUT (flag indicating maneuvering target detected);

     CLEAR  output flag;

     IF (range GE range threshold)
          THEN;      < separation negates need for maneuvering threat check>
     ELSEIF (vertical separation GE separation threshold)
          THEN;      < separation negates need for maneuvering threat check>
     ELSEIF (either AC's velocity LT velocity threshold)
          THEN;      <at least one AC slow so other tests can handle situation>
     OTHERWISE compute angle between aircraft paths;
               IF (aircraft paths are not parallel)
                    THEN;   <tau test can handle situation so no maneuver check>
                    ELSE PERFORM AC_path_comparison; <are AC offset and turning?>
                         IF (maneuvering target detected)
                              THEN SET output flag;

END AC_PARALLEL_OFFSET_TURNING_DETERMINATION;
```

```
--------------------------------------------------------------------------------
ROUTINE AC_PARALLEL_OFFSET_TURNING_DETERMINATION
   IN (GROUP INGROUP)      < group of thresholds >
   OUT (OUTFLAG);

     CLEAR (OUTFLAG, GOTHT);

     IF (ELENTRY.RANGE2 GE INGROUP.range_threshold)
          THEN:     < no maneuvering threat >
     ELSEIF (ELENTRY.ALT GE INGROUP.separation_threshold)
          THEN:
     ELSEIF ((SVECT1.VSQ LT INGROUP.velocity_threshold) OR
               (SVECT2.VSQ LT INGROUP.velocity_threshold))
          THEN:

     OTHERWISE COSA2 = ((SVECT1.XD*SVECT2.XD + SVECT1.YD*SVECT2.YD)**2) /
                    (SVECT1.VSQ * SVECT2.VSQ);
               IF (COSA2 LT INGROUP.cosine_thresh)
                    THEN:     < not parallel >
                    ELSE PERFORM AC_path_comparison;
                         IF (GOTHT = $TRUE)
                              THEN SET OUTFLAG;

END AC_PARALLEL_OFFSET_TURNING_DETERMINATION;
```

```
--------------------------------------------------------------------------------
PROCESS AC_path_comparison;

     <Checks the paths of both AC to decide if a dangerous geometry exists>

     Make AC1 first AC to check;

     REPEAT UNTIL(both AC checked or maneuvering status determined);
          Perform aircraft-dependent computations;
          IF (this AC parallel intrail with respect to other AC)
             <ie. is angle between position vector connecting 2 AC and
              AC velocity vector small ?>
               THEN not a maneuvering target threat;
          ELSEIF (this AC turning towards the other)
               THEN SET maneuvering target indication;
          access unchecked AC's state vector;
     ENDREPEAT;

END AC_path_comparison;
```

```
------------------------------------------------------------------------------------
PROCESS AC_path_comparison;

     PTR (TSVECT, OSVECT);     < 'target', 'other' state vector >
     CLEAR EXITLOOP;

     CLEAR DONEBOTH;
     TSVECT = SVECT1;     < arbitrarily make AC1 target >

     REPEAT UNTIL(EXITLOOP = $TRUE);
          OSVECT = state vector of non-target AC;
          RXVS = RY * OSVECT.XD - RX * OSVECT.YD;
          IF (AC2 is the target)
               THEN RXVS = -RXVS;
          IF (ELENTRY.RANGE2 GE INGROUP.zerodivide_thresh)
               THEN SINB2 = (RXVS**2) / (ELENTRY.RANGE2 * OSVECT.VSQ);
               ELSE SINB2 = INGROUP.sine_threshold;     < prevent zerodivide >
          IF (SINB2 LT INGROUP.sine_threshold)
               THEN SET EXITLOOP;     < Parallel intrail >
          ELSEIF ((RXVS GT 0 AND TSVECT.TURN = $STRNGLFT) OR
                  (RXVS LE 0 AND TSVECT.TURN = $STRNGRGT))
               THEN SET(GOTHT, EXITLOOP);
          ELSEIF (DONEBOTH = $TRUE)
               THEN SET EXITLOOP;     < we've looked at both >
          OTHERWISE TSVECT = OSVECT;     < get other state vector >
                    SET DONEBOTH;
     ENDREPEAT;

END AC_path_comparison;
```

```
--------------------------------------------------------------------------------
ROUTINE AIRCRAFT_PAIR_EQUIPMENT_AND_CONTROL_STATE_DETERMINATION
   IN  (state vectors of both AC)
   OUT (variables describing control & equipment state of pair);

     Determine equipage and control status;

END AIRCRAFT_PAIR_EQUIPMENT_AND_CONTROL_STATE_DETERMINATION;
```

```
--------------------------------------------------------------------------------

ROUTINE AIRCRAFT_PAIR_EQUIPMENT_AND_CONTROL_STATE_DETERMINATION
   IN  (SVECT1,SVECT2)
   OUT (PRCONT,PREQ);

     <Determines equipment status and control status of AC pair>

     PRCONT = $ONECONT;
     IF (SVECT1.CUNC EQ $TRUE AND SVECT2.CUNC EQ $TRUE)
          THEN PRCONT = $BOTHCONT;
     IF (SVECT1.CUNC EQ $FALSE AND SVECT2.CUNC EQ $FALSE)
          THEN PRCONT = $NOCONT;
     PREQ = $ONEEQ;
     IF ((SVECT1.ATSEQ EQ $AEQ OR SVECT1.ATSEQ EQ $ABEQ)
          AND (SVECT2.ATSEQ EQ $AEQ OR SVECT2.ATSEQ EQ $ABEQ))
          THEN PREQ = $BOTHEQ;
     IF (SVECT1.ATSEQ EQ $UNEQ AND SVECT2.ATSEQ EQ $UNEQ)
          THEN PREQ = $NOEQ;

END AIRCRAFT_PAIR_EQUIPMENT_AND_CONTROL_STATE_DETERMINATION;
```

```
--------------------------------------------------------------------------------

ROUTINE AREA_TYPE_DETERMINATION
   IN (aircraft characteristics)
   OUT (area_dependent variables);

     <On entry, AC's area type is zero (indicating "unknown")>

     Initialize controller alert request flag;

     IF (this AC is far from the sensor)
          THEN set AC area type to 4;

     IF (this AC passes area coarse screening)
          THEN REPEAT WHILE (Type 1 areas remain AND AC area type unknown);
                    Get next Type 1 area;
                    IF (aircraft is in this area)
                         THEN set AC area type to 1;
                              Save area index;     < unique identifier >
                              Save airfield name;
                              Save area's controller alert flag;
               ENDREPEAT;

               REPEAT WHILE (Type 2 areas remain AND AC area type unknown);
                    Get next Type 2 area;
                    IF (aircraft is in this area)
                         THEN set AC area type to 2;
                              Save area index;     < unique identifier >
                              Save airfield name;
                              Save area's controller alert flag;
               ENDREPEAT;

     IF (AC area type unknown)
          THEN set AC area type to 3;

END AREA_TYPE_DETERMINATION;

-------------------------- DETECT TASK HIGH-LEVEL LOGIC --------------------------
```

```
--------------------------------------------------------------------------------

ROUTINE AREA_TYPE_DETERMINATION
   INOUT (IOSVECT);

     SET IOSVECT.CAREQ;     < assume not inhibiting CA's >

     IF ((IOSVECT.X**2 + IOSVECT.Y**2) GT (DETPARM.RDIST**2))
          THEN IOSVECT.ACAT = $AT4;
     IF (IOSVECT.X LE AZPARM.ZHMXX AND IOSVECT.X GE AZPARM.ZHMNX AND
         IOSVECT.Y LE AZPARM.ZHMXY AND IOSVECT.Y GE AZPARM.ZHMNY)
          THEN LOOPIX = 1;
               REPEAT WHILE (LOOPIX LE AZPARM.NOI AND IOSVECT.ACAT = $UDAT);
                    Get next Type 1 area;
                    IF (aircraft is in this area)
                         THEN IOSVECT.ACAT = $AT1;
                              IOSVECT.IND = area index;     < unique identifier >
                              IOSVECT.ASSOC = airport name;
                              IOSVECT.CAREQ = area's controller alert flag;
                         ELSE LOOPIX = LOOPIX + 1;
               ENDREPEAT;
               LOOPIX = 1;

               REPEAT WHILE (LOOPIX LE AZPARM.NOII AND IOSVECT.ACAT = $UDAT);
                    Get next Type 2 area;
                    IF (aircraft is in this area)
                         THEN IOSVECT.ACAT = $AT2;
                              IOSVECT.IND = area index;     < unique identifier >
                              IOSVECT.ASSOC = airport name;
                              IOSVECT.CAREQ = area's controller alert flag;
                         ELSE LOOPIX = LOOPIX + 1;
               ENDREPEAT;

     IF (IOSVECT.ACAT = $UDAT)
          THEN IOSVECT.ACAT = $AT3;

END AREA_TYPE_DETERMINATION;
```

```
--------------------------------------------------------------------------------
ROUTINE ENCOUNTER_AREA_TYPE_DETERMINATION
   IN  (state vectors of AC, i.e., location of AC)
   OUT (encounter area of AC pair, flag to indicate if pair in zone2);

     <Finds area type and zone for each AC and combines area type for
      both AC into one index, ENAT, the encounter area type, used to
      control ATARS sensitivity>

     IF (AC1's area undetermined)
          THEN CALL AREA_TYPE_DETERMINATION;
     IF (AC1's zone undetermined)
          THEN CALL FINAL_APPROACH_ZONE_DETERMINATION;

     IF (AC2's area undetermined)
          THEN CALL AREA_TYPE_DETERMINATION;
     IF (AC2's zone undetermined)
          THEN CALL FINAL_APPROACH_ZONE_DETERMINATION;

     IF (both AC along glide slope for same airfield)
          THEN SET flag indicating situation;

     IF (either aircraft far from the sensor)
          THEN set encounter type to 4;
     ELSEIF (either aircraft in general vicinity of sensor)
          THEN set encounter type to 3;
     ELSEIF (either aircraft along active runway final approach)
          THEN IF (both AC near the same airfield)
                    THEN set encounter type to 2;
                    ELSE set encounter type to 3;
     ELSEIF (both aircraft in immediate vicinity of same airfield)
          THEN set encounter type to 1;
     OTHERWISE set encounter type to 2;

 END ENCOUNTER_AREA_TYPE_DETERMINATION;
```

```
-----------------------------------------------------------------------------------

ROUTINE ENCOUNTER_AREA_TYPE_DETERMINATION
   IN  (SVECT1, SVECT2)
   OUT (ENAT, INFAZ2);

     IF (SVECT1.ACAT EQ $UDAT)
          THEN CALL AREA_TYPE_DETERMINATION
                                  INOUT (SVECT1);
     IF (SVECT1.FAZ EQ $UDFAZ)
          THEN CALL FINAL_APPROACH_ZONE_DETERMINATION
                                  INOUT (SVECT1);
     IF (SVECT2.ACAT EQ $UDAT)
          THEN CALL AREA_TYPE_DETERMINATION
                                  INOUT (SVECT2);
     IF (SVECT2.FAZ EQ $UDFAZ)
          THEN CALL FINAL_APPROACH_ZONE_DETERMINATION
                                  INOUT (SVECT2);
     IF ((SVECT1.FAZ EQ $FAZ2 AND SVECT2.FAZ EQ $FAZ2) AND
        (SVECT1.ZPRT EQ SVECT2.ZPRT))
          THEN SET INFAZ2;
          ELSE CLEAR INFAZ2;
     IF (SVECT1.ACAT EQ $AT4 OR SVECT2.ACAT EQ $AT4)
          THEN ENAT = $AT4;
     ELSEIF (SVECT1.ACAT EQ $AT3 OR SVECT2.ACAT EQ $AT3)
          THEN ENAT = $AT3;
     ELSEIF (SVECT1.ACAT EQ $AT2 OR SVECT2.ACAT EQ $AT2)
          THEN IF (SVECT1.ASSOC EQ SVECT2.ASSOC)
                    THEN ENAT = $AT2;
                    ELSE ENAT = $AT3;
     ELSEIF (SVECT1.ASSOC EQ SVECT2.ASSOC)      < same Area 1 >
          THEN ENAT = $AT1;
     OTHERWISE ENAT = $AT2;

END ENCOUNTER_AREA_TYPE_DETERMINATION;
```

```
ROUTINE FINAL_APPROACH_ZONE_DETERMINATION
   IN (AC characteristics)
   OUT (zone_dependent variables);

     Set AC's zone to 0;

     IF (this AC passes zone coarse screening)
          THEN REPEAT WHILE (Type 1 zones remain AND AC's zone EQ 0);
                    Get next Type 1 zone;
                    IF (zone is active AND AC is in this zone)
                         THEN set AC's zone to 1;
                              Save airfield name;
               ENDREPEAT;

               REPEAT WHILE (Type 2 zones remain AND AC's zone EQ 0);
                    Get next Type 2 zone;
                    IF (zone is active AND AC is in this zone)
                         THEN set AC's zone to 2;
                              Save airfield name;
               ENDREPEAT;

END FINAL_APPROACH_ZONE_DETERMINATION;
```

```
ROUTINE FINAL_APPROACH_ZONE_DETERMINATION
   INOUT (IOSVECT);


     IOSVECT.FAZ = $FAZ0;


     IF (IOSVECT.X LE AZPARM.ZJMXX AND IOSVECT.X GE AZPARM.ZJMNX AND
         IOSVECT.Y LE AZPARM.ZJMXY AND IOSVECT.Y GE AZPARM.ZJMNY)
          THEN LOOPIX = 1;


               REPEAT WHILE (LOOPIX LE AZPARM.NOZ1 AND IOSVECT.FAZ = $FAZ0);
                    Get next Type 1 zone;
                    IF (zone is active AND AC is in this zone)
                         THEN IOSVECT.FAZ = $FAZ1;
                              IOSVECT.ZPRT = airfield name;
                         ELSE LOOPIX = LOOPIX + 1;
               ENDREPEAT;


               LOOPIX = 1;


               REPEAT WHILE (LOOPIX LE AZPARM.NOZ2 AND IOSVECT.FAZ = $FAZ0);
                    Get next Type 2 zone;
                    IF (zone is active AND AC is in this zone)
                         THEN IOSVECT.FAZ = $FAZ2;
                              IOSVECT.ZPRT = airfield name;
                         ELSE LOOPIX = LOOPIX + 1;
               ENDREPEAT;

END FINAL_APPROACH_ZONE_DETERMINATION;
```

```
--------------------------------------------------------------------------------

ROUTINE MINIMUM_APPROACH_DISTANCE_PREDICTION;

     Calculate miss distance;

END MINIMUM_APPROACH_DISTANCE_PREDICTION;
```

```
--------------------------------------------------------------------------------

ROUTINE MINIMUM_APPROACH_DISTANCE_PREDICTION
   IN  (VRX,VRY)
   OUT (MD2);

     FLT MD2;   <Predicted minimum horizontal miss distance for AC pair>

     VR2 = VRX**2 + VRY**2;

     IF (VR2 LT DETPARM.VMDTH)
          THEN MD2 = ELENTRY.RANGE2;
          ELSE MD2 = (RX*VRY - RY*VRX)**2 / VR2;

END MINIMUM_APPROACH_DISTANCE_PREDICTION;
```

```
--------------------------------------------------------------------------------------
ROUTINE RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS
   IN  (resolution advisory thresholds)
   OUT (flag to be set);     < either IFRFLG or CMDFLG >

     <Determines if Resolution Advisory is necessary>

     IF (AC violating horizontal resolution envelope)   < horizontal tests >
          THEN indicate horizontal proximity;     < immediate override >
     ELSEIF (AC will violate horizontal resolution envelope soon)
          THEN:     < passed horizontal tests >
     OTHERWISE failed horizontal tests;

     IF (AC violating vertical resolution envelope presently)     < vertical tests >
          THEN indicate vertical proximity;
     ELSEIF (AC will be coaltitude soon)
          THEN:     < passed vertical tests >
     OTHERWISE failed vertical tests;

     IF (passed all tests so far)
          THEN PERFORM vertical_divergence_filter;
                 <decide if RA won't be given as AC won't be violating
                   horizontal and vertical envelopes simultaneously>
               IF (pair did not pass filter)
                    THEN failed filter test;

     IF (all tests passed)
          THEN SET output flag to indicate RA required; < IFRFLG or CMDFLG >
               PERFORM proximity_checks;
         <determine if immediate RA required, ie, bypass 2/3 logic>

END RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS;
```

```
---------------------------------------------------------------------------------------

ROUTINE RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS
   IN (GROUP INGROUP)      < group of thresholds >
   OUT (OUTFLAG);

     BIT (NORES, FILTFAIL);
     CLEAR (NORES, OUTFLAG, HPROX, VPROX);

     IF (ELENTRY.RANGE2 LT INGROUP.range_threshold * RAPARM.BZP2) < horizontal tests>
          THEN SET HPROX;      < immediate override >
     ELSEIF (ELENTRY.TH LT INGROUP.hor_Tau_thresh)
          THEN ;
     OTHERWISE SET NORES;

     IF (ELENTRY.ALT LT INGROUP.immed_alt_thresh * RAPARM.BZP)      < vertical tests >
          THEN SET VPROX;
     ELSEIF (ELENTRY.TV GE 0 AND ELENTRY.TV LE INGROUP.vert_Tau_thresh)
          THEN ;
     OTHERWISE SET NORES;

     IF (NORES EQ $FALSE)
          THEN PERFORM vertical_divergence_filter;
               IF (FILTFAIL EQ $TRUE)
                    THEN SET NORES;

     IF (NORES EQ $FALSE)
          THEN SET OUTFLAG;      < IFRFLG or CHDFLG >
               PERFORM proximity_checks;

END RESOLUTION_TAU_AND_PROXIMITY_COMPARISONS;
```

```
--------------------------------------------------------------------------------
PROCESS vertical_divergence_filter;

    <This process decides if the resolution advisory should be inhibited
     if the vertical and horizontal minimum miss distances do not occur
     within a predefined time period>

    Compute absolute vertical range rate;

    IF (AC diverging vertically)
        THEN IF (AC are converging horizontally)
                THEN calculate vertical separation at time of
                                        minimum horizontal separation)
                    IF (calculated vertical separation outside RA envelope)
                        THEN inhibit RA;

END vertical_divergence_filter;
```

```
--------------------------------------------------------------------------------

PROCESS vertical_divergence_filter;

     CLEAR FILTFAIL;

     IF (ELENTRY.TV GE 0) <AC converging vertically>
          THEN: <do not perform test>
     ELSEIF (THTRU LE 0)  <AC diverging horizontally>
          THEN: <do not perform test>
     OTHERWISE TH = MIN(THTRU, RAPARM.DVDPT);
               TZ1 = SVECT1.ZD * TH;
               TZ2 = SVECT2.ZD * TH;
               TVMD = ABS(TZ2 - TZ1);
               IF (TVMD GT INGROUP.IMMED_ALT_THRESHOLD)
                 <IMMED_ALT_THRESHOLD will be AIFR or AF depending on INGROUP>
                    THEN SET FILTFAIL; <RA not wanted>

END vertical_divergence_filter;
```

```
--------------------------------------------------------------------------------
PROCESS proximity_checks:

    <If the AC are extremely close physically or temporally the 2/3
     window is bypassed and an immediate resolution advisory is given>

    IF (vertical proximity AND horizontal proximity)
        THEN indicate immediate resolution advisory needed;
        ELSE compute Tau minimums;
            IF (vertical proximity exists AND horizontal Tau LT minimum)
                THEN indicate immediate resolution advisory needed;
            ELSEIF (horizontal proximity exists AND vertical Tau LT minimum)
                THEN indicate immediate resolution advisory needed;
            ELSEIF (vertical Tau LT minimum AND horizontal Tau LT minimum)
                THEN indicate immediate resolution advisory needed;

END proximity_checks;
```

```
--------------------------------------------------------------------------------------

PROCESS proximity_checks;

     IF (VPROX EQ $TRUE AND HPROX EQ $TRUE)
          THEN SET ELENTRY.NTTFLG;
          ELSE AH = INGROUP.hor_Tau_thresh - (SYSTEM.SCANTM * RAPARM.TTM);
               AV = INGROUP.vert_Tau_thresh - (SYSTEM.SCANTM * RAPARM.TTM);
               IF (VPROX AND ELENTRY.TH LT AH)
                    THEN SET ELENTRY.NTTFLG;
               ELSEIF (HPROX AND ELENTRY.TV LT AV AND ELENTRY.TV GT 0)
                    THEN SET ELENTRY.NTTFLG;
               ELSEIF (ELENTRY.TH LT AH AND ELENTRY.TV LT AV
                                                  AND ELENTRY.TV GT 0)
                    THEN SET ELENTRY.NTTFLG;

END proximity_checks;
```

```
--------------------------------------------------------------------------------
ROUTINE TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION;

    <Determine variables used as thresholds throughout DETECT Task>

    IF (at least one AC controlled)
        THEN determine controller alert Tau thresholds;
            IF (AC not in immediate vicinity of airfield)
                THEN set remaining controller alert variables;

    IF (both AC controlled)
        THEN set threat and resolution advisory thresholds
                using c/c index in table 8-4, 8-5;
            Set resolution advisory Tau thresholds (UNC) to corresponding
                                                          CTL thresholds;

    ELSEIF (only one AC controlled)
        THEN set threat and resolution advisory thresholds
                using c/u index in tavle 8-4, 8-5;
            IF (controlled AC significantly faster than uncontrolled AC)
                THEN set threat and resolution advisory thresholds
                        for controlled AC to uncontrolled values;

    OTHERWISE CALL UNCON/UNCON_INDEX_DETERMINATION; <determine UUIND>
            Set threat and resolution advisory thresholds using UUIND
                and u/u index in table 8-4, 8-5;

END TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION;
```

```
--------------------------------------------------------------------------------

ROUTINE TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION
   IN  (ENAT, MULT, PREQ, PRCONT)
   INOUT (VRZA);


     IF (PRCONT NE $NOCONT)
          THEN CAVBL.TCONH = PDVBL.TWARN - ((PDVBL.RCONTH * R) /  <using table 8-3>
                                                  ELENTRY.DOT);    <for pdvbl values>
               VRZA = MIN(VRZA, THRSPARM.VRZCON);
               CAVBL.TCONV = PDVBL.TWARN - (PDVBL.ACONTH / VRZA);
               IF (ELENTRY.ENAT NE $AT1)
                    THEN set remaining controller alert variables in table 8-2;
     IF (PRCONT EQ $BOTHCONT)
          THEN set GROUP TAVBL.ctl_thresholds as defined in table 8-5;
               Set GROUP RAVBL.ctl_thresholds as defined in table 8-4;
               RAVBL.TCMDH = RAVBL.TIFRH;
               RAVBL.TCMDV = RAVBL.TIFRV;
     ELSEIF (PRCONT EQ $ONECONT)
          THEN set  GROUP TAVBL.ctl_thresholds as defined in table 8-5;
               Set  GROUP TAVBL.unc_thresholds as defined in table 8-5;
               Set  GROUP RAVBL.ctl_thresholds as defined in table 8-4;
               Set  GROUP RAVBL.unc_thresholds as defined in table 8-4;
               VRAT = VSQ(controlled_AC) / VSQ(uncontrolled_AC);
               IF (VRAT GT THRSPARM.VRATC)
                    THEN set GROUP TAVBL.ctl_thresholds to TAVBL.UNC;
                         Set GROUP RAVBL.ctl_thresholds to RAVBL.UNC;
     OTHERWISE CALL UNCON/UNCON_INDEX_DETERMINATION
                    IN  (MULT, PREQ)
                    OUT (UUIND);
                      Set GROUP TAVBL.unc_thresholds using table 8-5;
                      Set GROUP RAVBL.unc_thresholds using table 8-4;

END TAU_AND_PROXIMITY_THRESHOLD_DETERMINATION;
```

```
--------------------------------------------------------------------------------

ROUTINE THREAT_TAU_AND_PROXIMITY_COMPARISONS
  IN  (thresholds)     < for either controlled or uncontrolled >
  OUT (flag to be set);     < FPIFLG or FPWFLG >

    <Determines if Threat Advisory is necessary>

    IF ( AC outside horizontal threat envelope)
         THEN IF (AC won't violate horizontal threat envelope soon)
                   THEN failed horizontal test;

    IF (AC outside vertical threat envelope)
         THEN IF (AC won't be coaltitude soon)
                   THEN failed vertical test;

    IF (predicted minimum horizontal separation outside threat envelope)
         THEN failed miss distance test;

    IF (all tests passed)
         THEN SET output flag to indicate RA required;

END THREAT_TAU_AND_PROXIMITY_COMPARISONS;
```

```
-------------------------------------------------------------------------------------

ROUTINE THREAT_TAU_AND_PROXIMITY_COMPARISONS
   IN (GROUP INGROUP)      < group of thresholds >
   OUT (OUTFLAG);

     CLEAR (NOTHREAT, OUTFLAG);

     IF (ELENTRY.RANGE2 GE INGROUP.range_threshold)
          THEN IF (ELENTRY.TH GE INGROUP.hor_Tau_threshold)
                    THEN SET NOTHREAT;

     IF (ELENTRY.ALT GE INGROUP.separation_threshold)
          THEN IF (ELENTRY.TV GT INGROUP.vert_Tau_threshold)
                    THEN SET NOTHREAT;

     IF (ELENTRY.MD2 GE INGROUP.MD_threshold)
          THEN SET NOTHREAT;

     IF (NOTHREAT = $FALSE)      < all test passed >
          THEN SET OUTFLAG;

END THREAT_TAU_AND_PROXIMITY_COMPARISONS;
```

```
--------------------------------------------------------------------------------

ROUTINE UNCON/UNCON_INDEX_DETERMINATION;

     Determine UUIND index based on multiplicity, equipage, and speed ratio;

END UNCON/UNCON_INDEX_DETERMINATION;
```

```
---------------------------------------------------------------------------------

ROUTINE UNCON/UNCON_INDEX_DETERMINATION
     IN  (MULT, PREQ)
     OUT (UUIND);

     FLT UUIND;    <Index for uncont/uncont AC pair to be used in table look-up>
     FLT VRAT;     <Temporary local variable>
     INT TWO;      <Local constant = 2  >
     INT MULTIAC; <Number of AC in multiaircraft conflict (4) >

     IF (MULT GE MULTIAC)
          THEN UUIND = TWO;
     ELSEIF (PREQ EQ $NOEQ OR PREQ EQ $BOTHEQ)
          THEN UUIND = 1;
     OTHERWISE VRAT = VSQ(equipped_AC) / VSQ(unequipped_AC);
               IF (VRAT LT PDPARM.VRATTH)
                    THEN UUIND = TWO;
                    ELSE UUIND = 1;

END UNCON/UNCON_INDEX_DETERMINATION;
```

9. TRAFFIC ADVISORY TASK

9.1 Purpose

The Traffic Advisory Task generates and stores data used in traffic advisory messages to equipped aircraft. When the Detect Task (Section 8) finds that a pair qualifies for traffic advisories or ATCRBS Track Blocks, Detect Task sets appropriate flags in the Encounter List entry.

ATARS must generate a unique set of messages for every equipped aircraft. For example, the aircraft in an encounter pair may differ in their equipage or level of ATARS service. In the latter case, different forms of messages are assembled for the two aircraft. The Traffic Advisory Task examines each subject aircraft in sequence and saves data appropriate to the aircraft's class of service.

9.2 The PWILST

The data structure used to store traffic and other non-resolution advisories is the PWILST. This is a linked list of entries for each aircraft, accessed from the aircraft State Vector. The list may contain any or all of the following types of entry: Traffic Advisory (Proximity or Threat); Terrain, Airspace, Obstacle Warnings (T/A/O); ATCRBS Track Blocks (TB); Altitude Echo (ALEC). The PWILST structure was introduced in Section 3.3.2, and in Pseudocode Section 3.4. Only an ATARS-equipped aircraft will ever have Traffic Advisory or ALEC entries on its list. Only aircraft equipped with ATARS and BCAS will ever have ATCRBS TB entries. However, an unequipped aircraft may have T/A/O warnings generated if it is under ATC control, as ATC may uplink these warnings.

The PWILST is created an entry at a time. Traffic Advisory entries are created as Encounter List pairs are processed. Entries are kept and are updated on subsequent scans. Each entry specifies the aircraft (or obstacle) for which it is providing a warning, so that duplicates are avoided. All entries are sorted into groups and ordered within each group by the Data Link Message Construction (DLMC) Task (Section 16.2.1). To facilitate this ordering, certain ranking data is calculated by the Traffic Advisory Task, while the Encounter List entry is still available.

Certain entries are created by other tasks. The ALEC entry may be created by either the DLMC Task or the Report Processing Task. T/A/O entries are created by the task of the same name (Section 6.3).

The deletion of PWILST entries is discussed in Section 16.2.4.

9.3 Traffic Advisory Entries

ATARS sends an aircraft all of its traffic advisory messages from one site. When an aircraft is in a seam (see Section 6.2), the site designated primary sends these messages. However, ATCRBS TB messages may be duplicated by sites serving the pair of aircraft.

9.4 Pseudocode for Traffic Advisory Task

The pseudocode for Traffic Advisory Task follows. The Match Routine is used to search for a previous entry containing the same object aircraft. If one is found, its data is updated. In this case, the entry type may change from Proximity to Threat or vice versa. If no matching object aircraft entry is found, a new one is created and inserted in the PWILST. When creating a new entry, the track number field must be set to an uninitialized value, the END field cleared, and OLD_TYPE set to "none."

PSEUDOCODE TABLE OF CONTENTS

```
---------------------------------------------------------------------------

STRUCTURE TAPARM

GROUP msg_format
     FLT ALT_EXT_INCR             <increment for altitude extension>
     FLT ALT_EXT_LIM              <limit for altitude extension>
     FLT CLOCK_INCR               <increment for clock position>
     FLT COURSE_INCR              <increment for heading>
     FLT FINE_BRG_INCR            <increment for fine bearing>
     FLT FINE_HDG_INCR            <increment for fine heading>

GROUP ranking
     FLT LARGET                   <large value of tau for divergence>

ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
---------------------------------------------------------------------------------

STRUCTURE TAVBL

   GROUP identity
     PTR SUBJECT                <pointer for own aircraft>
     PTR OBJECT                 <pointer for other aircraft>
     BIT MATCHED                <matching PWILST entry found>
     INT TEMP_TYPE              <temporary value of entry type>

   GROUP calculations
     FLT A                      <intermediate calculation>
     FLT BEARING                <bearing to other aircraft>
     FLT DOTP                   <used in calc. of range rate>
     FLT HEADING                <heading of other aircraft>
     FLT RNKTAU                 <tau used for ranking entries>
     FLT RX                     <relative x-coordinates>
     FLT RXP                    <projected relative x-coord.>
     FLT RY                     <relative y-coodinates>
     FLT RYP                    <projected relative y-coord.>
     FLT RZP                    <projected relative z-coord.>

ENDSTRUCTURE;
```

```
--------------------------------------------------------------------------------
TASK TRAFFIC_ADVISORY
   IN (pairs from encounter list)
   OUT (PWILST entries);

     <Scan all encounter list entries. For each equipped aircraft,
      generate traffic advisory and ATCRBS_TB entries as required
      and link onto aircraft's PWILST structure.>

     REPEAT WHILE (more pairs on encounter list);
       Select encounter list entry;
       subject=ac1;
       object=ac2;
       LOOP:
          IF (subject ATARS equipped);
               THEN IF (not primary for subject AND other site sees subject)
                         THEN delete all TA entries from PWILST;
                    ELSEIF (entry shows TA Threat status)
                         THEN PERFORM proximity_data_calculation;
                              PERFORM threat_data_calculation;
                              CALL MATCH; <update PWILST entry or create new one>
                    ELSEIF (entry shows TA proximity status)
                         THEN PERFORM proximity_data_calculation;
                              CALL MATCH;
                    OTHERWISE; <no TA flags set or not primary>
                    IF (BCAS flag set in encounter list)
                         THEN PERFORM ATCRBS_track_block_calculation;
       EXITIF (subject=ac2);
          subject=ac2;
          object=ac1;
       ENDLOOP;

     ENDREPEAT;

END TRAFFIC_ADVISORY;

--------------------- TRAFFIC ADVISORY TASK HIGH-LEVEL LOGIC ---------------------
```

```
----------------------------------------------------------------------------------
TASK TRAFFIC_ADVISORY
     <scan all encounter list entries. For each equipped aircraft,
       generate traffic advisory and ATCRBS_TB entries as required
       and link onto aircraft's PWILST structure.>
   IN (pairs from encounter list)
   OUT (PWILST entries);
     REPEAT WHILE (more pairs on encounter list);
       Select next ELENTRY;
       SUBJECT=ELENTRY.ACID1;
       OBJECT=ELENTRY.ACID2;
       LOOP;
          IF (SUBJECT->SVECT.ATSEQ EQ $AEQ or $ABEQ)
               THEN IF (SUBJECT->SVECT.PSTAT EQ $FALSE AND any other site bit
                    set in SVECT.GEOG)
                         THEN delete all TA_PROX and TA_THREAT  entries from PWILST;
                    ELSEIF (ELENTRY.TAREQ AND (ELENTRY.FPIFLG OR ELENTRY.FPWFLG) set)
                         THEN PERFORM proximity_data_calculation;
                              PERFORM threat_data_calculation;
                              CALL MATCH
                                IN (TA entry, object AC ID)
                                INOUT (subject AC PWILST);
                    ELSEIF (ELENTRY.TAREQ AND NOT (ELENTRY.FPIFLG OR ELENTRY.FPWFLG) )
                         THEN PERFORM proximity_data_calculation;
                              CALL MATCH
                                IN (TA entry, object AC ID)
                                INOUT (subject AC PWILST);
                    OTHERWISE; <no TA flags set or not primary>
                    IF (ELENTRY.BOFFREQ set)
                         THEN PERFORM ATCRBS_track_block_calculation;
       EXITIF (SUBJECT EQ ELENTRY.ACID2)
          SUBJECT=ELENTRY.ACID2;
          OBJECT=ELENTRY.ACID1;
       ENDLOOP;
     ENDREPEAT;
END TRAFFIC_ADVISORY;
--------------------- TRAFFIC ADVISORY TASK LOW-LEVEL LOGIC ----------------------
```

```
--------------------------------------------------------------------------------

PROCESS proximity_data_calculation:

   <Compute data for TA message according to AC class of ATARS service (TA_class).>

     Compute clock bearing;
     Compute relative altitude;  <special code indicates altitude unknown>
     Compute range;
     Compute course of object AC;

     Compute weighted range;
     Save negative in rank data;  <smallest range ranks highest>

     IF (TA_class GT 0)
          THEN Compute fine bearing;
               Save object AC ATC control state;
               Save object AC ATARS equipage state;

     IF (TA_class GT 1)
          THEN Compute object AC groundspeed;
               Save object AC climb performance;
               Save object AC abbreviated data;

END proximity_data_calculation;
```

```
--------------------------------------------------------------------------------------

PROCESS proximity_data_calculation;

            <compute data for TA msg according to AC TA_class of service>
                <function INT(x) means take integer part of (x+0.5)>
          <suffix 1 means SUBJECT->SVECT.  suffix 2 means OBJECT->SVECT. >
     RX = X2-X1;
     RY = Y2-Y1;
     A = RX*XD1 + RY*YD1;
     BEARING = ARC COS (A/SQRT((ELENTRY.RANGE2)*(XD1**2 + YD1**2)));
     Correct BEARING for proper quadrant;
     TA_PROX.CLOCK_BRG = INT(BEARING/CLOCK_INCR);
     IF (TA_PROX.CLOCK_BRG EQ 0)
        THEN TA_PROX.CLOCK_BRG = 12;
     TA_PROX.REL_ALT = Z2 - Z1;
     TA_PROX.RANGE = SQRT(ELENTRY.RANGE2);
     HEADING = ARC COT(YD2/XD2);
     TA_PROX.COURSE = INT(HEADING/COURSE_INCR);
     Correct TA_PROX.COURSE for proper quadrant;
     IF (TA_PROX.COURSE = 8)
        THEN TA_PROX.COURSE = 0;
     TA_PROX.RANGE_WEIGHTED = Two's complement of
                               RX**2 + RY**2 + SYSTEM.VWEIGHT* (Z2-Z1)**2 ;
     TA_PROX.TAU = 0;

     IF (SUBJECT-> SVECT.ACLASS NE SCL0)
        THEN TA_PROX.FINE_BRG=INT((BEARING-(CLOCK_BRG-0.5)*CLOCK_INCR)/FINE_BRG_INCR);
             TA_PROX.CONTROL = OBJECT-> SVECT.CUNC;
             TA_PROX.ATARS_EQP = OBJECT-> SVECT.ATSEQ;

     IF (SUBJECT-> SVECT.ACLASS EQ SCL2)
        THEN TA_PROX.GRND_SPEED = SQRT(XD2**2 + YD2**2);
             TA_PROX.CLIMB_PERF = OBJECT-> SVECT.ACLP;
             TA_PROX.ABBREV = OBJECT-> SVECT.ACAB;

END proximity_data_calculation;
--------------------- TRAFFIC ADVISORY TASK LOW-LEVEL LOGIC -----------------------
```

```
-------------------------------------------------------------------------------
PROCESS threat_data_calculation;

   <Compute data for threat msg according to AC TA_class of service.>

     IF (maneuvering target flag set)
          THEN Compute pseudo-tau from range and velocities;
                      <a worst-case tau if the pair turns head-on>
     ELSEIF (the pair is diverging)
          THEN Tau=large constant;
     OTHERWISE Use horizontal tau from Detect task;
     Save negative of tau in rank data;<small tau gets highest rank>

     IF (TA_class EQ 0)
          THEN Save object AC ATC control state;
               save object AC ATARS equipage state;
          ELSE compute predicted horiz. miss distance;  <tracked data only>
               Compute object AC vertical speed;      <using tracked data only>
               Compute relative altitude extension field;
               Compute fine heading field;
               IF (strong turn delared for object aircraft)
                    THEN Save object aircraft turn status;
                    ELSE Indicate object aircraft not turning;

END threat_data_calculation;
```

```
--------------------------------------------------------------------------------------
PROCESS threat_data_calculation;

     Change entry TYPE to TA_THREAT;
     Change TA_PROX fields to corresponding TA_THREAT fields;
     IF (ELENTRY.HTTFLG set)
          THEN RNKTAU = ELENTRY.RANGE2/(ELENTRY.ACID1->SVECT.VSQ
                                 + ELENTRY.ACID2->SVECT.VSQ);
     ELSEIF (ELENTRY.DOT GT $ZERO)
          THEN RNKTAU = LARGET;
     OTHERWISE RNKTAU = ELENTRY.TH;
     TA_THREAT.TAU = Two's complement of RNKTAU;

          <compute data for threat msg according to AC TA_class of service>
     IF (SUBJECT-> SVECT.ACLASS EQ $CL0)
          THEN TA_THREAT.CONTROL = OBJECT-> SVECT.CUNC;
               TA_THREAT.ATARS_EQP = OBJECT-> SVECT.ATSEQ;
          ELSE TA_THREAT.HMD = SQRT(ELENTRY.HD2);
               TA_THREAT.VERT_SPD = ZD2;
               IF (ABS(REL_ALT) LT ALT_EXT_LIM)
                    THEN TA_THREAT.REL_ALT_EXT = 0;
                    ELSE TA_THREAT.REL_ALT_EXT =
                              INT((ABS(REL_ALT)-ALT_EXT_LIM)/ALT_EXT_INCR);
               TA_THREAT.FINE_HDG =
                         INT((HEADING - (COURSE-0.5)*COURSE_INCR)/FINE_HDG_INCR);
               IF (OBJECT->SVECT.TURN EQ $STRNGLFT OR $STRNGRGT)
                    THEN TA_THREAT.TURN = OBJECT->SVECT.TURN;
                    ELSE TA_THREAT.TURN = $STRAIGHT;

END threat_data_calculation;
```

```
--------------------------------------------------------------------------------

PROCESS ATCRBS_track_block_calculation;


    Set subject AC = the BCAS-equipped AC;
    Set object AC = the other AC;
    END field = 0;
    Project both aircraft straight ahead for one scan;

    Compute range in nmi;
    Compute range_rate in knots;
    Compute bearing from subject aircraft <BCAS-equipped> to
       object aircraft <ATCRBS> in degrees measured positive clockwise from north;
             <bearing refers to aircraft positions only, not headings>
    Obtain projected altitude of object aircraft in feet;
    Obtain vertical rate of object aircraft in ft/s;
    Type = ATCRBS_TB entry;

    CALL MATCH;     <Store range, range_rate, bearing, altitude, and vertical rate
                                  in PWILST entry>

END ATCRBS_track_block_calculation;
```

```
---------------------------------------------------------------------------------------
PROCESS ATCRBS_track_block_calculation;


     IF (ELENTRY.ACID1-> SVECT.ATSEQ EQ $ABEQ)
          THEN SUBJECT = ELENTRY.ACID1;
               OBJECT = ELENTRY.ACID2;
          ELSE SUBJECT = ELENTRY.ACID2;
               OBJECT = ELENTRY.ACID1;
          <suffix 1 means SUBJECT->SVECT.  suffix 2 means OBJECT->SVECT. >
     CLEAR ATCRBS_TB.END;
     RXP = XP2 - XP1;
     RYP = YP2 - YP1;
     RZP = ZP2 - ZP1;
     ATCRBS_TB.RANGE = SQRT(RXP**2 + RYP**2 + RZP**2);
     DOTP = RXP*(XD2-XD1) + RYP*(YD2-YD1) + RZP*(ZD2-ZD1);
     ATCRBS_TB.RANGE_RATE = DOTP/ATCRBS_TB.RANGE;
     ATCRBS_TB.BEARING = ARC TAN (RXP/RYP);
     Correct ATCRBS_TB.BEARING for proper quadrant;
     ATCRBS_TB.ALT = Z2 + ZD2*SYSTEM.SCANT;
     ATCRBS_TB.VERT_RATE = ZD2;
     Type = ATCRBS_TB entry;

     CALL MATCH
       IN (ATCRBS_TB entry with object AC ID)
       INOUT (subject AC PWILST);

END ATCRBS_track_block_calculation;
```

```
ROUTINE MATCH
   IN (TA entry with object AC ID)
   INOUT (subject AC PWILST);

    <Search PWILST, update old entry for object AC or create new entry.>

     REPEAT WHILE (entries on subject PWILST of type desired);
                                   <type TA or ATCRBS_TB>
          IF (entry ID=object AC ID)
               THEN found match;
               ELSE;
     EXITIF (found match); <pointing to matched entry>
          Find next entry;
     ENDREPEAT;
     IF (found match)
          THEN Save old type;
               Replace entry data with new entry;   <type may change>
          ELSE Link new entry;  <prox or threat at top of PWILST,
                                              ATCRBS_TB at bottom>
               Store new entry data;
               Old type=none;   <only applies to prox or threat>

END MATCH;
```

```
--------------------------------------------------------------------------------

ROUTINE MATCH
   IN (TA entry with object AC ID)
   INOUT (subject AC PWILST);
          <search PWILST, update old entry for obj. AC or create new entry>

     CLEAR MATCHED;
     REPEAT WHILE (entries on subject PWILST of type desired);
                                  <type TA or ATCRBS_TB>
          IF (entry OBJ_AC EQ OBJECT)
               THEN SET MATCHED;
               ELSE;
     EXITIF (MATCHED set); <pointing to matched entry>
          Find next PWILST entry;
     ENDREPEAT;
     IF (MATCHED set)
          THEN TEMP_TYPE = old type;
               Replace entry data with new entry;   <type may change>
               OLD_TYPE=TEMP_TYPE;
          ELSE Link new entry;  <Prox or threat at top of PWILST,
                                             ATCRBS_TB at bottom>
               Store new entry data;
               OLD_TYPE=none;   <only applies to Prox or threat>

END MATCH;
```

10. SEAM PAIR TASK

10.1 Purpose

A set of responsibility rules is used to prevent duplicate action by neighboring sites. This task determines the site's responsibility to perform resolution or controller alert for each pair on the Encounter List. When resolution is to be performed, this task creates a Pair Record if none exists. If any connected sites are giving service to either aircraft, this task sends a message to these sites to claim the pair for resolution. The task also determines whether resolution pairs are to be processed by Master Resolution (Normal) Task or require ground line coordination first, before Master Resolution (Delayed) Task. If conditions have changed so that own-site is no longer responsible for resolution, resolution deletion status is indicated.

10.2 Site Responsibility

ATARS resolution responsibility is in most cases assigned to a single site for each individual aircraft pair. If an equipped aircraft is in a seam (Section 6.2), that aircraft may receive resolution advisories from different sites due to different threats, or in certain cases, due to a single ATCRBS-equipped threat. As an aircraft flies into or out of sites' coverage areas, those sites' responsibilities may change. The same is true when a site drops service for an aircraft due to lack of target reports. Finally, when an ATARS site sees a conflict pair, it always has priority over BCAS for resolution of the conflict.

The responsibility rules are complex, to account for numerous combinations of aircraft equipage and site-to-site connectivity. These rules are applied on a pairwise basis between sites. Where three or four sites see the same pair, the rules ensure a single site taking responsibility.

1. <u>ATCRBS - ATCRBS Conflict</u>

   Own-site is responsible for controller alert.

2. <u>DABS - DABS Conflict</u>

   a. For initial resolution, own-site is responsible if highest common site ID of both aircraft is own-ID. If connected sites see either aircraft, responsible site sends claim message to establish Pair Record at other site showing that site responsible.

b. On subsequent scans, if Pair Record exists and shows a connected site in charge, own-site is not responsible.

c. If Pair Record exists and shows handoff, site uses highest ID rule as in (2a) to decide own responsibility.

d. If Pair Record exists and shows a non-connected site in charge, own-site responsible if own-ID is higher than that site's ID.

e. If no Pair Record exists but both aircraft are receiving advisories from a common ATARS site, then (2a) does not apply. If no site with a higher ID than own-ID sees both aircraft and is giving resolution advisories to both, then own-site is responsible. Otherwise, own-site is not responsible. This test is repeated every scan, for as long as own-site sees a need for resolution or controller alert.

3. <u>DABS - ATCRBS Conflict With Both Aircraft Above Altitude ALTDC</u>

a. If no Pair Record exists containing the DABS, or a Pair Record exists and indicates handoff, then if own-ID is the highest site ID set in the DABS aircraft, own-site is responsible. If a higher ID site sees the DABS, own is not responsible.

b. If a Pair Record exists for the pair and shows a connected site responsible, then own is not responsible.

c. If any Pair Record exists containing the DABS which shows a non-connected site responsible whose ID is greater than own-ID, then own is not responsible.

d. If (3c) does not apply and a Pair Record shows own-site responsible, then own-site remains responsible.

e. If all Pair Records containing the DABS show lower ID, non-connected sites responsible, then own-site is responsible.

The effect of (3) for non-connected sites is to create a "hand-off" using site ID's when an aircraft flies from a low-ID site's interior into a seam. When in the seam, the low-ID site keeps sending resolution using rule (3d), while the higher site begins sending resolution using rule (3e). After the low-ID site reads down the other's resolution and creates another Pair Record, it applies rule (3c) and removes its resolution. (See Resolution Deletion Task, Section 15).

4. DABS - ATCRBS Conflict With One or Both Aircraft Below Altitude ALTDC

   a. If no Pair Record exists or the Pair Record indicates handoff status, then if the DABS has no resolutions from another site (excluding any from the site handing off responsibility) or if any resolution advisories are from a lower-ID site, then own-site is responsible. If the DABS has any resolution advisory from a higher ID site (excluding handoff), then own-site is provisionally responsible.

   b. If a Pair Record shows a connected site in charge, then own-site is not responsible.

   c. If a Pair Record shows a non-connected site in charge, then responsibility is determined by the source of the DABS resolutions as in (4a).

The status "provisionally responsible" is indicated in the Encounter List entry to signal Master Resolution Task (Section 12) to test the higher site's resolution advisory. If it appears adequate, own-site will not complete resolution; otherwise own-site will resolve.

10.3 Pseudocode for Seam Pair Task

The pseudocode for Seam Pair Task follows. The local variable O_ID is used to hold the correct own-site ID to use for each pair tested. This is always equal to the SYSTEM.OWNID except while operating in the Backup-Master mode (Section 17.3.2), where a different ID is sent to aircraft in the Center zone.

PSEUDOCODE TABLE OF CONTENTS

```
--------------------------------------------------------------------------------

STRUCTURE SEAMPARM
  GROUP miscellaneous
     FLT ALTDC                           <altitude of good dual sensor coverage>

ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
------------------------------------------------------------------------------

STRUCTURE SEAMVBL
  GROUP miscellaneous
     BIT RESP                          <own site responsible for resolution>
     INT TEST_ID                       <temp. storage for testing various site IDs>
     INT O_ID                          <value of own-site ID to use for this pair>

ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
--------------------------------------------------------------------------------
TASK SEAM_PAIR
   IN (state vectors)
   OUT (messages to connected sites)
   INOUT (sector encounter list, conflict tables);

     <Determine site resolution responsibility for pairs. Create pair records.>
     REPEAT WHILE (more pairs on encounter list);
       Select next encounter list entry;
       IF (resolution status OR controller alert status)
         THEN PERFORM responsibility:
              IF (own site not responsible)
                THEN Reset Controller Alert, Resolution status flags;
                     IF (pair record exists AND shows own site)
                         THEN Indicate resolution deletion status;
                ELSE IF (resolution status)
                         THEN IF (no pair record)
                                   THEN Create pair record;
                                   <If DABS-ATCRBS pair, DABS is AC1>
                                        IF (ac1,ac2 are in separate confl. tables)
                                           THEN Merge conflict tables:
                                                Seam flag = logical OR
                                                  of seam flags in old tables;
                                        IF (either AC seen by connected site(s) )
                                           THEN Send CLAIM msgs to connected site(s);
                                                SET seam flag;
                              ELSEIF (pair record shows another site in charge)
                                   THEN Store own site in charge;
                                        Send CLAIM msgs to connected site(s);
                              OTHERWISE;   <pair record shows own site in charge>
                              IF (seam flag set in conflict table)
                                   THEN Indicate delayed resolution;
                                   ELSE Indicate normal resolution;
     ENDREPEAT;

END SEAM_PAIR;
------------------------ SEAM PAIR TASK HIGH-LEVEL LOGIC --------------------------
```

```
--------------------------------------------------------------------------------------
TASK SEAM_PAIR
   IN (state vectors, SYSVAR)
   OUT (messages to connected sites)
   INOUT (sector encounter list, conflict tables);

     REPEAT WHILE (more pairs on encounter list);
          Select next ELENTRY;
          IF (ELENTRY.RAREQ set OR ELENTRY.CNAREQ set)
            THEN PERFORM responsibility;
                 IF (RESP EQ $FALSE)
                   THEN CLEAR ELENTRY.RAREQ and ELENTRY.CNAREQ;
                        IF (pair rec. exists AND PREC.ATSID EQ O_ID)
                          THEN SET ELENTRY.RDREQ;
                   ELSE IF (ELENTRY.RAREQ set)
                          THEN IF (no pair record)
                                 THEN Create pair record;
                                 <If DABS-ATCRBS pair, make DABS AC1>
                                      IF (ELENTRY.ACID1->SVECT.CTPTR
                                            NE ELENTRY.ACID2->SVECT.CTPTR)
                                        THEN Merge conflict tables;
                                             CTHEAD.SEAM = logical OR
                                                 of CTHEAD.SEAM's in old tables;
                                      IF (either AC SVECT.GEOG contains
                                                 connected site(s) )
                                        THEN Send CLAIM messages to connected site(s);
                                             SET CTHEAD.SEAM;
                               ELSEIF (PREC.ATSID NE O_ID)
                                 THEN PREC.ATSID = O_ID;
                                      Send CLAIM messages to connected site(s);
                               OTHERWISE:   <pair record shows own site in charge>
                               IF (CTHEAD.SEAM set)
                                 THEN SET ELENTRY.DELREQ;
                                 ELSE;< normal resolution >
     ENDREPEAT;
END SEAM_PAIR;
------------------------ SEAM PAIR TASK LOW-LEVEL LOGIC ------------------------
```

```
--------------------------------------------------------------------------------
PROCESS responsibility;

                  <Determine whether own site is responsible for pair.>

     IF (Backup-master mode)
          THEN Determine Own_ID to use for testing this pair;

     IF (ac1 is ATCRBS) AND (ac2 is ATCRBS)
          THEN own responsible;  <for controller alert only>
     ELSEIF (ac1 is DABS) AND (ac2 is DABS)
          THEN PERFORM DABS-DABS_responsibility;
     OTHERWISE <DABS-ATCRBS conflict>
               IF (both AC high altitude)
                    THEN PERFORM high_altitude_DABS-ATCRBS_responsibility;
                    ELSE PERFORM low_altitude_DABS-ATCRBS_responsibility;

END responsibility;
```

```
--------------------------------------------------------------------------------
PROCESS responsibility:

    IF (SYSVAR.MASTER EQ $FALSE)
        THEN O_ID=SYSTEM.OWNID;
    ELSEIF (AC1->SVECT.CENTR set AND AC2->SVECT.CENTR set)
        THEN O_ID=SYSVAR.FAILED;  <use failed site ID in Center zone>
    ELSEIF (AC1->SVECT.CENTR not set AND AC2->SVECT.CENTR not set)
        THEN O_ID=SYSTEM.OWNID;
    OTHERWISE O_ID=higher of SYSVAR.FAILED or SYSTEM.OWNID;
                                 <conflict split between own & Center>

    IF (ELENTRY.ACID1->SVECT.TYPE EQ $ATCRBS AND
          ELENTRY.ACID2->SVECT.TYPE EQ $ATCRBS)
        THEN SET RESP;  <for controller alert only>
    ELSEIF (ELENTRY.ACID1->SVECT.TYPE EQ $DABS AND
              ELENTRY.ACID2->SVECT.TYPE EQ $DABS)
        THEN PERFORM DABS-DABS_responsibility;
    OTHERWISE <DABS-ATCRBS conflict>
        IF (ELENTRY.ACID1->SVECT.Z GT ALTDC AND ELENTRY.ACID2->SVECT.Z GT ALTDC)
            THEN PERFORM high_altitude_DABS-ATCRBS_responsibility;
            ELSE PERFORM low_altitude_DABS-ATCRBS_responsibility;

END responsibility;
```

```
--------------------------------------------------------------------------------------
PROCESS DABS-DABS_responsibility;

    IF ((no pair record exists) AND (both aircraft not receiving
      res. adv. from any common site)) OR (handoff bit set in pair record)
        THEN determine highest common site ID;
            IF (highest=own_ID)
                THEN own responsible:
                ELSE own not responsible;
    ELSEIF (pair record exists) AND (site shown is not own_ID AND is connected)
        THEN own not responsible;

    ELSEIF (pair record exists) AND (site shown is not own_ID)
       <can get pair record for non-connected site thru connected site>
        THEN IF (that site's ID GT own_ID)
                THEN own not responsible;
                ELSE own responsible;

    OTHERWISE test_id = highest possible ID;
            own responsible;
            REPEAT WHILE (test_id GT own_ID);
            EXITIF (own not responsible)
                IF (test_id is not a connected site AND sees both aircraft
                        AND test_id is giving res. adv. to both aircraft)
                    THEN own not responsible;
                    ELSE;
                decrement test_id;
            ENDREPEAT;

END DABS-DABS_responsibility;
```

```
PROCESS DABS-DABS_responsibility;

     IF ((no pair record exists) AND (both aircraft not receiving
       res. adv. from any common site)) OR (PREC.HDOFF set)
          THEN determine highest common site ID;
               IF (highest=O_ID)   <OWNID or failed site_ID if set>
                    THEN SET RESP;
                    ELSE CLEAR RESP;
     ELSEIF (pair record exists) AND (PREC.ATSID NE O_ID
                                            AND is connected site)
          THEN CLEAR RESP;

     ELSEIF (pair record exists) AND (PREC.ATSID NE O_ID)
        <can get pair rec. for non-connected site thru connected site>
          THEN IF (PREC.ATSID GT O_ID)
                    THEN CLEAR RESP;
                    ELSE SET RESP;

     OTHERWISE SET RESP;
               TEST_ID = highest possible ID;
               REPEAT WHILE (TEST_ID GT O_ID);
                    IF (TEST_ID is not a connected site AND is set in both
                              SVECT.GEOG1 and GEOG2
                              AND both AC in pair recs with PREC.ATSID EQ TEST_ID)
                         THEN CLEAR RESP;
                         ELSE;
               EXITIF (RESP not set);
                    decrement TEST_ID;
               ENDREPEAT;

END DABS-DABS_responsibility;
```

```
PROCESS high_altitude_DABS-ATCRBS_responsibility;

     Search for pair records for this pair or with this DABS vs. unknown AC;
     IF (no such pair rec exists OR handoff set in all such pair recs)
          THEN <the DABS is not receiving any res. advisory for threat>
               IF (own_ID is highest in DABS)
                    THEN own responsible;
                    ELSE own not responsible;

          ELSE <pair record exists and isn't handoff>
               IF (site in pair record not own_ID AND is connected)
                    THEN own not responsible;
               ELSEIF (site in any such pair record GT own_ID)
                    THEN own not responsible;
                    ELSE own responsible; <own already resolving or take over
                                                   from lower ID site>

END high_altitude_DABS-ATCRBS_responsibility;
```

```
--------------------------------------------------------------------------------
PROCESS high_altitude_DABS-ATCRBS_responsibility;

     Search for pair records for this pair or this DABS vs. a PREC.PAC=$ONE;
     IF (no such pair rec exists pair OR PREC.HDOFF set in all such pair recs)
          THEN <the DABS is not receiving any res adv. for threat>
               IF (O_ID is highest bit set in SVECT.GEOG of AC
                         whose SVECT.TYPE EQ $DABS)
                    THEN SET RESP;
                    ELSE CLEAR RESP;

          ELSE <pair record exists and PREC.HDOFF not set>
               IF (pair rec found with PREC.ATSID NE O_ID AND is connected site)
                    THEN CLEAR RESP;
               ELSEIF (PREC.ATSID GT O_ID in any such pair record)
                    THEN CLEAR RESP;
                    ELSE SET RESP;

END high_altitude_DABS-ATCRBS_responsibility;
```

```
--------------------------------------------------------------------------------------
PROCESS low_altitude_DABS-ATCRBS_responsibility;

    IF (no pair record exists for pair OR handoff bit set in pair record)
        THEN IF (DABS has no resolution advisory from other site,
                 excluding site handing off)
               THEN own responsible;
             ELSEIF (no DABS resolution advisory is from site higher
                     than own_ID, excluding site handing off)
               THEN own responsible;
               ELSE own provisionally responsible;

    ELSEIF (pair record shows connected site)
        THEN own not responsible;
    ELSEIF (no DABS resolution advisory is from non-connected site
               higher than own_ID)
        THEN own responsible;
        ELSE own provisionally responsible;

END low_altitude_DABS-ATCRBS_responsibility;
```

```
-------------------------------------------------------------------------------------
PROCESS low_altitude_DABS-ATCRBS_responsibility;

     IF (no pair record exists for pair OR PREC.HDOFF set)
          THEN IF (DABS has no resolution advisory from other site,
                     excluding site handing off)
                    THEN SET RESP;
               ELSEIF (no DABS resolution advisory is from site higher
                     than O_ID, excluding site handing off)
                    THEN SET RESP;
                    ELSE SET RESP and ELENTRY.RAPROV;

     ELSEIF (PREC.ATSID EQ a connected site)
          THEN CLEAR RESP;
     ELSEIF (no DABS resolution advisory is from non-connected site
                  higher than O_ID)
          THEN SET RESP;
          ELSE SET RESP;
               SET ELENTRY.RAPROV;

END low_altitude_DABS-ATCRBS_responsibility;
```

11. CONTROLLER ALERT PROCESSING

Two tasks are covered in this section, the Conflict Resolution Data (CRD) Task and the Resolution Notification (RN) Task. Respectively, their functions are delivery of the Conflict Resolution Data Message and delivery of the Resolution Notification Message. These messages are forms of the Controller Alert Message defined in Section 5.1.3.2. The Conflict Resolution Data Message tells a controller that an aircraft pair will receive ATARS resolution advisories if both aircraft remain on the same heading. This message contains a "preview" of the intended resolution advisory. This preview advisory is based on information available to ATARS at the moment of message generation. It is determined only once and does not reflect possible changes in ATARS resolution advisories based on aircraft course changes. This resolution data is only available to the controller.

The Resolution Notification Message is sent to the controller when one or both aircraft in a conflict pair have been delivered an ATARS resolution advisory. The message contains the actual advisory presently being displayed to the pilot. If the resolution advisory changes on a later scan, the content of the message changes appropriately.

11.1 Message Initiation

A Resolution Data Message is sent when the last three (CRDPARM.MCTA) of five (CRDPARM.NCTA) scans have had the controller alert flag set in the Detect Task (Section 8), or is sent immediately whenever the ICAFLG flag is set in the Detect Task. This message continues to be sent, each scan until a Resolution Notification Message is sent or the last three scans have had no controller alert flags set in Detect, at which point the data message is discontinued. The Resolution Notification Message is sent when any aircraft in the conflict pair has acknowledged reception of a resolution advisory.

11.2 Resolution Pair Acknowledgement and Controller Alert Lists

The above operation is accomplished by maintaining two lists, the Resolution Pair Acknowledgement List (RPALST) and Controller Alert List (CALIST). Once an encounter pair (from Detect) has ICAFLG or CAFLG set the first time, an entry is made on the (non-sectored) CA list. During each succeeding scan, an existing CA list pair is updated with either a null bit (when no ICA or CA flag set) or a set bit (when either CA or ICA flag is set) in the sliding window. A scan is determined by the passage of 10 seconds (CRDPARM.CRDSCAN) or the CA flags being set again in Detect. A Conflict Resolution Data Message is then sent if (1) the 3/5 window is satisfied or ICAFLG is set and (2) no Resolution Notification Message will be sent this scan or has been sent on previous scans. Deletions from CALIST only occur when an entry has timed out (no flags set in last 3 scans).

The Resolution Notification Message is simply sent whenever an entry appears in RPALST (i.e., acknowledgement has been received of a successful uplink of a resolution advisory to one or both aircraft). This list is used as a communication link between the RAR Task (which actually receives the acknowledgment) and the Resolution Notification Task. The RAR Task creates the entries, while the RN Task or CRD Task deletes the entries. An entry is deleted when both the CRD Task (which checks to see if an acknowledgment message is occurring this scan) and the RN Task (which sends the acknowledgment message) have acted upon the same pair or when the RPALST entry has not been updated for a sufficient time (times out). Entries are normally deleted before a time out condition occurs.

11.3 Message Format

In the pseudocode (Section 11.4) the instruction "send message" implies the message will be formatted according to Tables 11-1, 11-2, and 11-3 (all information required for message content is available at this point) and sent to an ATC receiving station. It is left to this station or stations to determine the dissemination of the message within ATC facilities, using the aircraft identification fields in the message.

11.4 Pseudocode for Controller Alert Processing

The pseudocode for the two tasks which comprise the Controller Alert Processing follow. The Conflict Resolution Data Task can run whenever there is an entry in the Encounter List and is dependent on the Detect and Seam Pair Tasks. The Resolution Notification Task can only operate after the RAR Task has finished updating the Resolution Pair Acknowledgement List.

TABLE 11-1

CONFLICT RESOLUTION DATA MESSAGE FORMAT[1]

BIT NUMBER	PARAMETER	DESCRIPTION
1-8	CA message identification	10000011
9-32	ACID1	AC1 DABS ID or ATCRBS code with surveillance file no.
33-56	ACID2	AC2 DABS ID or ATCRBS code with surveillance file no.
57-58	CS1	Control/equipment state AC1.
59-69	RA1	Resolution advisory for AC1. The code used to enter advisories is given in Table 11-3
70-72	DEL1	Delivery status for AC1, DEL1 = 001 (CRD Message)
73-74	CS2	Control/equipment state for AC2
75-85	RA2	Resolution advisory for AC2
86-88	DEL2	Delivery status for AC2, DEL2 = 001
89	V1	Not presently used
90	V2	Not presently used
91-92	AMTYP	Message type, AMTYP = 00 for resolution advisory
93-96		Spare

[1]The contents of Tables 11-1, 11-2, 11-3 are taken from Reference 8.

TABLE 11-2

RESOLUTION NOTIFICATION MESSAGE FORMAT

The resolution notification message format is the same as Table 11-1, except for the DEL field.

Possible values are:

DEL1 = 011, DEL2 = 011	Resolution advisory delivered to both aircraft
DEL1 = 011, DEL2 = 010	Resolution advisory delivered to AC1 only
DEL1 = 010, DEL2 = 011	Resolution advisory delivered to AC2 only

TABLE 11-3

RESOLUTION ADVISORY FIELD CODE

The Resolution Advisory Field is an 11-bit field in the Controller Alert Message which signifies the Resolution Advisory that is being delivered to the aircraft for execution. A double-dimension advisory has both of the appropriate bits set. The field is encoded as follows:

BIT POSITION IN 11-BIT FIELD	RESOLUTION ADVISORY INDICATED
1	Turn right
2	Turn left
3	Climb
4	Descend
5	Don't turn right
6	Don't turn left
7	Don't climb
8	Don't descend
BITS 9, 10, 11 OF 11-BIT FIELD	
000	No vertical speed limit
001	Limit climb to 500 ft/min
010	Limit climb to 1000 ft/min
011	Limit climb to 2000 ft/min
100	Limit descent to 500 ft/min
101	Limit descent to 1000 ft/min
110	Limit descent to 2000 ft/min

PSEUDOCODE TABLE OF CONTENTS

```
--------------------------------------------------------------------------------

STRUCTURE CRDPARM
   GROUP ovrhd
     INT MCTA         <number of bits in sliding window which causes
                       message to be sent >
     INT NCTA         <size of window in bits >
     INT CRDSCAN      <CRD message refresh time , i.e., time after which
                       a zero entry is made in sliding window >
     INT RN_TIME_OUT  <entries still remaining after RN_TIME_OUT
                       seconds are deleted from the RPA list >
ENDSTRUCTURE;
```

PRECEDING PAGE BLANK-NOT FILMED

```
-----------------------------------------------------------------------------

TASK CONFLICT_RESOLUTION_DATA
   IN  (encounter list entries requiring CA processing,
          controller alert list, CRDVBL, associated state
          vectors, resolution pair acknowledgment list (RPA list))
   OUT (conflict resolution data message (CRD),
         updated CA list, updated RPA list);

     <Controls the generation of the conflict resolution data message.
      This message informs the controller of a resolution advisory that
      will be sent to an AC pair if no evasive action occurs>

     PERFORM cull_RPA_list;  <delete all entries that have timed out>
     PERFORM update_CA_list; <delete timed out entries, slide window
                               with 0 fill if more than one scan has
                               elapsed, generate CRD message as needed>
     LOOP (on encounter list entries requiring CA processing for this
            sector);

          Get next entry on encounter list requiring CA processing;
     EXITIF (no more entries);

          IF (pair not on CA list)
               THEN create entry on list;
          Slide window and set last bit of window in CA list entry;
          Update time in CA list entry;
          IF (First time CRD message required for this pair)
               THEN PERFORM CRD_message_determination;
               ELSE: <do nothing, CRD not required or if required under control
                          of update_CA_list>
          Mark encounter entry as CA processed;

     ENDLOOP;

END CONFLICT_RESOLUTION_DATA;
```

-------------------- CONTROLLER ALERT TASKS HIGH-LEVEL LOGIC ----------------------

```
--------------------------------------------------------------------------------
TASK CONFLICT_RESOLUTION_DATA
   IN  (encounter list entries requiring CA processing,
        controller alert list, CRDPARM, SVECT1, SVECT2,
        resolution pair acknowledgment (RPA) list)
   OUT (Conflict Resolution Data Message,
       updated CA list, updated RPA list):

     PERFORM cull_RPA_list;  <delete all entries that have timed out>
     PERFORM update_CA_list; <delete timed out entries, slide window
                              with 0 fill if more than one scan has
                              elapsed, generate CRD message as needed>
     LOOP (on encounter list entries requiring CA processing for this
            sector);

          Select next entry on list;
     EXITIF (no more entries);

          IF (pair not on CA list)
               THEN create entry on list;
          Slide CALIST.WNDSTR and set last bit;
          CALIST.CATIME = present time;
          IF ((last CRDPARM.MCTA bits of CRDPARM.NCTA bits of CALIST.WNDSTR set
               OR ELENTRY.ICAFLG set) AND CALIST.CRDSNT not set)
               THEN PERFORM CRD_message_determination;
                    SET CALIST.CRDSNT;  <indicate CRD message initiated>
               ELSE; <do nothing, CRD not required or if required under control
                         of update_CA_list>
          CLEAR ELENTRY.CALREQ;

     ENDLOOP;

END CONFLICT_RESOLUTION_DATA;
```

```
--------------------------------------------------------------------------------

PROCESS cull_RPA_list;

     <Examines all entries on RPALST and removes timed out entries>

     LOOP (on all entries in RPA list);

          Get next entry on RPALST;
     EXITIF (no more entries);

          IF (this entry put on long time ago)
               THEN delete entry;

     ENDLOOP;

END cull_RPA_list;
```

```
--------------------------------------------------------------------------------

PROCESS cull_RPA_list;

    LOOP (on all entries in RPA list);

        Get next entry on RPALST;
    EXITIF (no more entries);

        IF ((SYSVAR.CTIME - RPALST.TIME) GT CRDPARM.RN_TIME_OUT)
            THEN delete entry;

    ENDLOOP;

END cull_RPA_list;
```

```
PROCESS update_CA_list;

    <Once CRD message is sent, it continues to be sent every 'scan'
     until (1) CA times out, or, (2) resoluion notification (RN)
     message is sent>

    LOOP (on all entries in CA list);

        Get next entry on CALIST;
    EXITIF (no more entries);

        IF (one scan has elapsed since last update)
            THEN slide window over 1 bit and zero fill;
                IF (entry has not timed out)
                    THEN update time in CA list entry;
                        IF (CRD message (tentatively) required);
                            THEN PERFORM CRD_message_determination;
                    ELSE delete entry;

    ENDLOOP;

END update_CA_list;
```

```
--------------------------------------------------------------------------------------

PROCESS update_CA_list;

<Once CRD message is sent, it continues to be sent every 'scan'
 until (1) CA times out, or, (2) BN message is sent>

    LOOP (on all entries in CA list);

        Get next entry on CALIST;
    EXITIF (no more entries on list);

        IF ((SYSVAR.CTIME - CALIST.CATIME) GT CRDPARM.CRDSCN)
            THEN slide window over 1 bit and zero fill;
                IF (last CRDPARM.MCTA bits of CALIST.WNDSTR not all zero)
                    THEN CALIST.CATIME = SYSVAR.CTIME;
                        IF (CALIST.CRDSNT EQ STRUE)
                            THEN PERFORM CRD_message_determination;
                    ELSE delete entry;

    ENDLOOP;

END update_CA_list;
```

```
--------------------------------------------------------------------------------
PROCESS CRD_message_determination;

     <Creates and sends a CRD message if no RN message has been sent
      or will be sent this scan>

     IF (RN message sent on previous scans )
          THEN; <do nothing, no CRD message required.for this pair>
     IF (pair on RPA list)
          THEN mark CALIST entry to indicate RN message sent or will be sent;
               IF (RN message sent this scan) <i.e., RN Task has sent message>
                    THEN delete RPA entry; <do nothing else as CRD not required>
                    ELSE denote CRD processing as done in RPA entry; <no CRD
                       message required as RN message will be sent this scan>
     ELSEIF (conflict resolutions not already determined) <no RPA entry for pair>
               THEN PERFORM compute_conflict_resolution_data;
          Send CRD preview message;

END CRD_message_determination;
```

```
--------------------------------------------------------------------------------

PROCESS CRD_message_determination;

     <Creates and sends a CRD message if no RN message has been sent
      or will be sent this scan>

     <On entering this process the CALIST entry for this pair has
      been located on list>

     IF (CALIST.ACKSNT EQ STRUE);
          THEN: <do nothing, no CRD message required for this pair>
     IF (pair on RPA list)      <must search RPALST>
          THEN SET CALIST.ACKSNT;   <prevent unnecessary RPALST searches>
               IF (RPALST.CARN EQ STRUE) <i.e., RN Task has  sent message>
                    THEN delete RPA entry;  <do nothing else as CRD not required>
                    ELSE SET RPALST.CACRD;  <no CRD message required as RN message
                                             will be sent this scan for pair>
     ELSE IF (CALIST.CRDSNT NE STRUE)
               THEN PERFORM compute_conflict_resolution_data;
          Send CA preview message;

END CRD_message_determination;
```

```
--------------------------------------------------------------------------------
PROCESS compute_conflict_resolution_data;

     <Generates resolution advisories to be inserted
      in CRD message for controller>

     IF (both AC unequipped)
          THEN set resolutions to null for both AC;
          ELSE IF (resolution not already generated for this pair)
                    THEN Define necessary input variables for PAER routine;
                         CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE;
                         set resolutions in CA list entry for this pair;
          OTHERWISE; do nothing as resolutions already entered;

END compute_conflict_ resolution_data;
```

```
--------------------------------------------------------------------------------
PROCESS compute_conflict_resolution_data;

     <Generates resolution advisories to be inserted
      in CRD message>

     IF (SVECT1.ATSEQ EQ $UNEQ AND SVECT2.ATSEQ EQ $UNEQ)
          THEN CALIST.RESH1 = $NULLRES;
               CALIST.RESV1 = $NULLRES;
               CALIST.RESH2 = $NULLRES;
               CALIST.RESV2 = $NULLRES;
          ELSE IF (CALIST.CRDSNT EQ $FALSE)
                    THEN CALL ASEP_computation;
                         SNGDIM = $TRUE;
                         MRCAP = $FALSE;
                         CALL RESOLUTION_ADVISORIES_EVALUATION_ROUTINE
                              IN  (ELENTRY, DUMPTR, ASEP, SNGDIM, MRCAP)
                              OUT (RADPTR);
                         CALIST.RESH1 = RADPTR -> RADS.H1;
                         CALIST.RESV1 = RADPTR -> RADS.V1;
                         CALIST.RESH2 = RADPTR -> RADS.H2;
                         CALIST.RESV2 = RADPTR -> RADS.V2;
          OTHERWISE; <do nothing resolutions already entered in CALIST>

END compute_conflict_resolution_data;
```

```
---------------------------------------------------------------------------------

TASK RESOLUTION_NOTIFICATION
   IN  (RPA list)
   OUT (updated RPA list, RN messages to controller);

     <Sends a RN message for all entries on RPALST and deletes this entry
      if CRD Task has run>

     LOOP (on RPA list):

          Get next entry on RPALST;
     EXITIF (no more entries on list);

          Send RN message to CA;
          IF (CONFLICT RESOLUTION DATA TASK has run)
               THEN delete entry; <both tasks have run, entry no longer needed>
               ELSE indicate that this task has sent RN message;

     ENDLOOP;

END RESOLUTION_NOTIFICATION;
```

```
--------------------------------------------------------------------------------

TASK RESOLUTION_NOTIFICATION
   IN  (RPA list)
   OUT (updated RPA list, RN messages to controller);

     LOOP (on RPA list);

          Get next entry on RPALST;
     EXITIF (no more entries on list);

          Send RN message to CA;
          IF (RPALST.CACRD EQ $TRUE)
               THEN delete entry;
               ELSE SET RPALST.CARN;

     ENDLOOP;

END RESOLUTION_NOTIFICATION;
```

*U.S. GOVERNMENT PRINTING OFFICE: 1981-341-428/1274--1-3